COMPUTER ALGEBRA ON MIMD MACHINE[1]

Jean-Louis ROCH, Pascale SENECHAUD
Françoise SIEBERT-ROCH, Gilles VILLARD

Algorithmique Parallèle et Calcul Formel, Laboratoire TIM3
Institut National Polytechnique de Grenoble
38031 GRENOBLE Cedex FRANCE

Abstract : PAC is a computer algebra system, based on MIMD type parallelism. It uses parallelism as a tool for processing problems wich are too complex for a sequential treatment. Basic fundamentals of the system are firstly discussed. Then, different problems are studied, particularly the implementation of infinite-precision arithmetic, the solution of linear systems and of Diophantine equations, the parallelization of Buchberger's algorithm for Gröbner bases.
A prototype of PAC is implemented on the Floating Point System hypercube Tesseract 20 (16 nodes), and different timing results obtained on this machine are given.

[1]This work is partially supported by the *PRC Mathématiques et Informatique* and by the *Greco Calcul Formel* of French *Centre Nationnal de la Recherche Scientifique.*

I / INTRODUCTION

I.1 / General Presentation of PAC

We are mainly interested in the parallelization of mathematical Computer Algebra algorithms : that is why the symbolic part is performed by a classical system (CAS) wich is installed on a host machine. This system may call parallel algorithms developped on a MIMD machine, directly connected to the host. Conversions between the different formats - on host and on the parallel machine - are performed by an interface. **PAC** may be also considered as an algebraic (soft) co-processor, associated to a Computer Algebra host system.

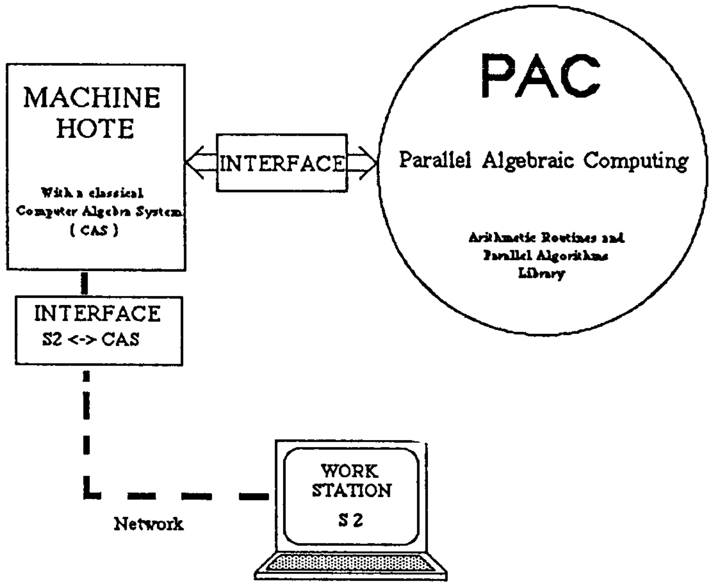

MACHINE HOTE
With a classical Computer Algebra System (CAS)
INTERFACE
PAC
Parallel Algebraic Computing
Arithmetic Routines and Parallel Algorithms Library
INTERFACE S2 <-> CAS
Network
WORK STATION S 2

Figure I.1 : General Overview of PAC

The first aim of PAC system is also to allow implementation of parallel algorithms, solving usual Computer Algebra problems on a MIMD machine, considering the parallel machine as a peculiar device specialized in algebraic operations processing.

In the following, we will describe :

*	H	: the host computer
*	CAS	: the host Computer Algebra System
*	P	: the parallel machine connected to H

Within the framework of the laboratory and of this study, H is a μVax II and P is the MIMD computer FPS Tesseract 20 (16 processors) [11][14].

This approach may be generalized : P may be considered as a big device accessible by any Computer Algebra System - on a work station - via a network. P would be used for large tasks that cannot be sequentially solved. The interface is then seen as a symbolic server.

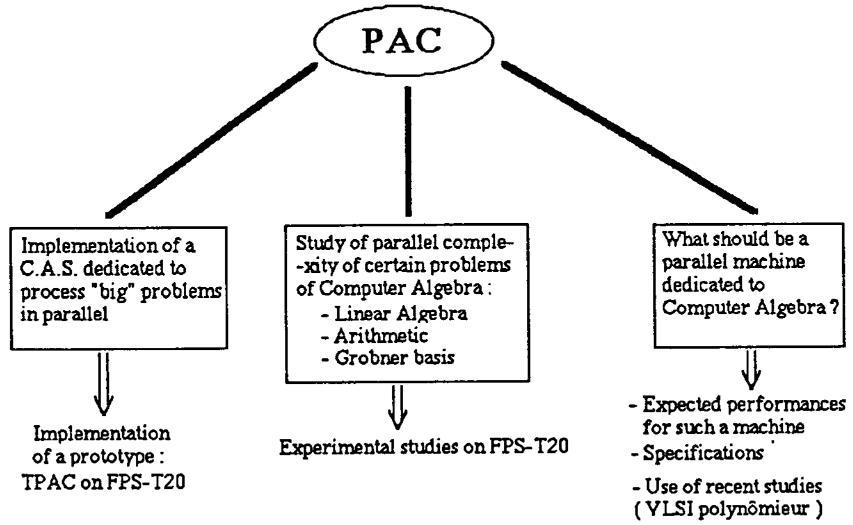
I.2 / Different application domains of PAC

Our study concerns essentially three different types of problems :

* Arithmetic and linear algebra classical problems : diophantine equations, linear systems resolution....

* Standard basis problems, and mainly the implementation of a parallel algorithm to find the Gröbner basis associated to a boolean polynomials family (with applications in circuits formal proof).

* The design of a specific machine dedicated to process algebra problems : most of the operations performed by a computer algebra system should be efficiently processed with a dedicated architecture.


PAC
Implementation of a C.A.S. dedicated to process "big" problems in parallel
Study of parallel comple- -xity of certain problems of Computer Algebra : - Linear Algebra - Arithmetic - Grobner basis
What should be a parallel machine dedicated to Computer Algebra ?
Implementation of a prototype : TPAC on FPS-T20
Experimental studies on FPS-T20
- Expected performances for such a machine
- Specifications
- Use of recent studies (VLSI polynômieur)

II / ARITHMETIC

II.1 / General Presentation

Structures chosen for objects - possibly recursive - representation, have to respect parallel constraints. Mainly, storage has to be such that communicate basic objects is easy. Saad has prooved [30] that the easiest structure to communicate is an array - with an hypercube topology - . That is why we have chosen to store integers and rationnals in special arrays (*blocks*) - managing memory with special routines- [2][25][28].

Different basic objects fields have been implemented, allowing :

* arithmetic operations on $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{Q}$

* arithmetic operations on $\mathbb{Z}[X]$ or $\mathbb{Q}[X]$

* calculations on boolean polynomials

II.2 / Arithmetic on $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{Q}$

Implementation

One of the major advantage of PAC is its portability : that is why all routines are written in C language.

Obviously, to increase the speed of some basic routines (like arithmetic calculations in $\mathbb{N}$), some modules have been rewritten in assembly language for Inmos T414 transputer. We have chosen to store data in array : this allows a good efficiency in data communication between processors ([28][30]). But, then, to allocate places in memory is sometimes very long, as merge or compaction of memory is necessary. To manage memory on each node, we use the C primitives *malloc* and *free*.

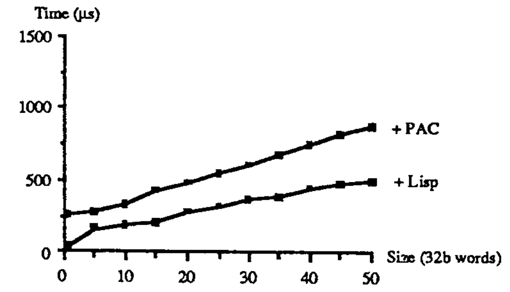
Integers are represented by decomposition in 2^{32} basis. This computation basis has been chosen because of the existence of an extended arithmetic on each transputer. However, it is possible to use other basis. Conversions in 10^9 basis are made by the user interface.
Rationals are stored in array - as the *concatenation* - of two integers.

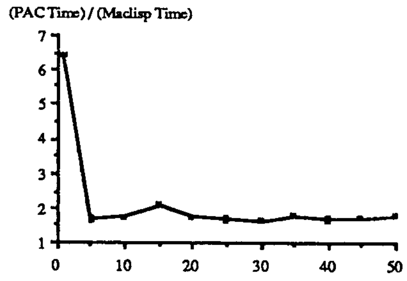
Algorithms

Addition, subtraction, multiplication and division are performed by classical carry-save algorithms ([21]). GCD is computed using Lehmer algorithm for large integers. A version of this algorithm computes Bezout algorithm to perform modular lifting.
Rational arithmetic is built on integer arithmetic.

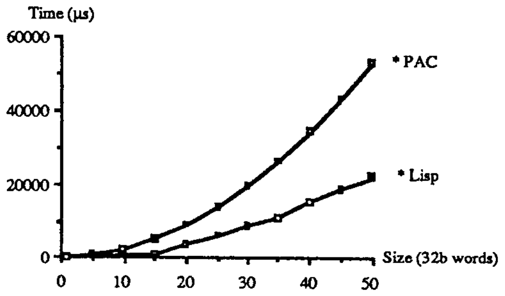
Performances

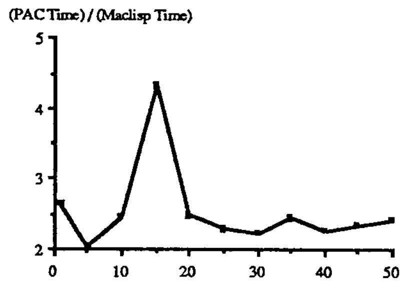
It's very difficult to have a good evaluation of efficiency of arithmetic operations, as there is no other infinite precision arithmetic system implemented on transputers. We chose to compare PAC performances to Maclisp. We use the version of Maclisp implented on Bull-DPS 8 (Multics system). We compare times of compiled (C) and interpreted (I) Lisp programs.
Those diagrams show that we obtain the same order as Maclisp for complexity, with a certain factor of proportion ($\simeq 2$). This remark is right for all arithmetic operations, except for division - this operation has not been yet optimized in assembly language-. But, comparisons time in Pac is very often constant (comparing the sizes and first words of both integers is often sufficient).
In fact, as the following diagrams prove it, this proportion factor comes essentially from the different capacities of the two machines : the T414 Transputer is based on a RISC architecture, while DPS8 is a CISC . For instance, it is impossible to use registers with the T414.



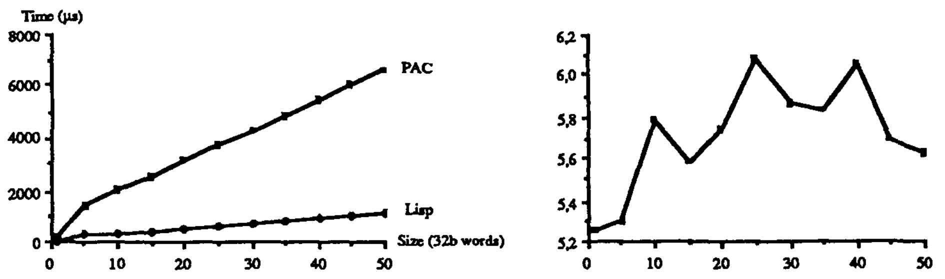


Addition

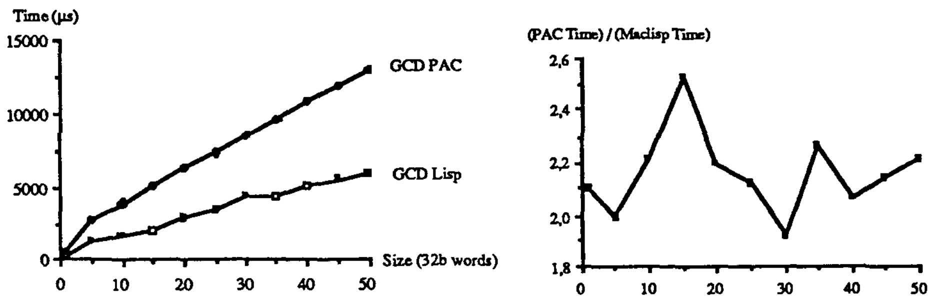




Multiplication



Division



Greatest Common Divisor

II.3 / Arithmetic on $\mathbb{Z}[X]$: Parallel Multiplication of Univariate Polynomials

Let

$$P = \sum_{i=0}^{n} p_i X^{e_i} \quad \text{and} \quad Q = \sum_{j=0}^{m} q_j X^{f_j}$$

If we consider the product as a sum of n+1 polynomials:

$$\sum_{i=0}^{n} p_i X^{e_i} * \sum_{j=0}^{m} q_j X^{f_j} = \sum_{i=0}^{n} \left(\sum_{j=0}^{m} p_i q_j X^{e_i + f_j}\right)$$

The implemented algorithm is a mere decomposition of the n sums. If the number of processors is p, each processor has to compute about n/p sums; then, by a lifting process, the results are summed two by two. We give some examples allowing for the comparison between communication and computation costs.

$P_0 = (x + 10!)\ \hat{}\ 20$ $\qquad$ $P_3 = (x^4 + x^3 + x^2 + x + 1)\ \hat{}\ 5$

$P_1 = P_0\ \hat{}\ 2$ $\qquad$ $P_4 = P_3\ \hat{}\ 2$

$P_2 = (x + 1)\ \hat{}\ 20$ $\qquad$ $P_5 = P_3\ \hat{}\ 3$

	0	1	2	3	4
$P_0 * P_0$	6673	4101	2864	2595	2693
$P_1 * P_0$	—	21639	12825	8883	7881
$P_0 * P_1$	—	21194	13499	11787	11778
$P_2 * P_2$	2949	1824	1360	1324	1465
$P_3 * P_5$	35363	17834	11349	9774	9804
$P_5 * P_3$	35801	17007	9208	6337	5570
$P_4 * P_4$	34325	14460	7610	5300	4833

Time in ms against hypercube dimension

For examples 2 and 3, on one processor, the coefficients size leads to memory overflow. From a certain number of processors, communication cost becomes too expensive with respect to the arithmetic costs; to obtain good performances, the calculus to perform has to be consequent in each processor.
It appears that for a sufficient number of processors it is more efficient to multiply the highest degree polynomial by the other one (more communication time) [see ex. 2-3 and 5-6].

III / LINEAR ALGEBRA

Some of the basic algebraic computation algorithms (such as Gaussian elimination over GF(p) presented below) can be easily developed from the corresponding parallel numerical algorithms, just doing little modifications. But most of them need totally new approaches. Even if an important amount of theoretic work has been done the implementation of those algorithms leads to new problems :

- the chosen architecture and a high grained parallelism model do not correspond to theoretic models using not bounded numbers of processors,
- in view of an implementation we have to take in account the constraints associated to the communications, which cost can be a major loss of time; for Gaussian elimination over GF(p) the communication cost represent from 1/10 to 1/3 of the total cost. So, implementing parallel algorithms, the main work will consist in minimizing the communication cost using as much as possible the inherent parallelism of the algorithms.

The first implemented algorithm concern the resolution of linear Diophantine equations and so the computation of n integers' gcd. We then present Gaussian elimination over GF(p) and the resolution of linear systems over integers. The first obtained results show that important sized problems (which would ask for days of computations on a simple sequential machine) have been successfully treated. This will permit us to consider more complex problems such as normal forms of matrices (Hermite's normal form) and reduction in lattices.

III.1 / Linear Diophantine Equations

We present here an algorithm described by W.A.Blankinship [5] determining the GCD d of n positive integers $a_1, a_2, \ldots, a_n$ and giving a solution $x_1, x_2, \ldots, x_n$ for the associated Diophantine equation:

$$d = a_1 x_1 + a_2 x_2 + \ldots + a_n x_n.$$

Let the equation :

$$(1) \qquad a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = b$$

with $a_1, a_2, \ldots a_n$, b integers. This equation has an integral solution if and only if the a_i's GCD divides b.

Theorem :
Let d be the GCD of the a_i ($i = 1, \ldots n$). Let us assume that $b = c\,d$ where c is a positive integer.Let $x_0 \in \mathbb{Z}^n$ be a particular solution of (1). It exists n - 1 independent solutions of the homogeneous equation $s_1, s_2, \ldots, s_{n-1} \in \mathbb{Z}^n$ and the general solution $x \in \mathbb{Z}^n$ of (1) is (see [32]):

$$(2) \qquad x = (x_0 * c) + \lambda_1 s_1 + \lambda_2 s_2 + \ldots + \lambda_{n-1} s_{n-1} \quad \text{with } \lambda_1, \lambda_2, \ldots, \lambda_{n-1} \in \mathbb{Z}$$

ALGORITHM DESCRIPTION

Let $D = (d_{ij})_{(i = 1 \text{ to } n,\, j = 1 \text{ to } n+1)}$ be the matrix n x (n + 1) $\begin{bmatrix} a_1\ 1\ 0 \ldots 0 \\ a_2\ 0\ 1 \ldots 0 \\ \ldots \\ a_n\ 0 \ldots 0\ 1 \end{bmatrix}$

The algorithm of W.A. Blankinship consist in performing elementary transformations on D rows until the first column contains no more than one non-zero coefficient. Let r and r' two rows of non zero leading coefficients . Let us assume that the two leading coefficients r_1 and r_1' are such that $r_1 \geq r_1' > 0$; the elementary row transformation applied is (see [20]) :

$$\begin{bmatrix} u & v \\ \frac{-r'_1}{g_0} & \frac{r_1}{g_0} \end{bmatrix} \begin{bmatrix} r \\ r' \end{bmatrix} = \begin{bmatrix} g_0 & x\ x \ldots x \\ 0 & x\ x \ldots x \end{bmatrix}$$

The only non-zero first column coefficient which is obtained after these operations is the a_i's gcd, and his right coefficients constitute a particular solution of the equation (see [24]). The n-1 others rows gives n-1 independant solutions for the homogeneous equation.

THE PARALLEL ALGORITHM

We try to use the inherent parallelism of the algorithm: gcd $(a_1, a_2, \ldots, a_n, \ldots, a_{2n})$ = gcd (d_1, d_2) where d_1= gcd $(a_1, \ldots, a_n)$ and d_2 = gcd = $(a_{n+1}, \ldots, a_{2n})$.

Parallel algorithm:
first step:
choice of a good sequential method on each processor and execution of the process on submatrices.
second step:
lifting process to perform elementary transformation on the rows with non-zero first column coefficient remaining in each processor.

The size of the manipulated objects increases fastly with the number of a_i and can lead to a memory overflow.
The parallel programming allows a manipulation of smaller matrices and so the total number of operations performed is less than in sequential.

P ALGORITHM (on $p = 2^{\text{dim}}$ processors)

```
SEQ
    $\alpha = n / p$ (here p is supposed to be a divisor of n)
    The label of processor is proc
    Inputs: each processor receives $\alpha$ positive integers
    processor n°proc = $(b_{dim-1}, \ldots, b_1, b_0)_2$ receives : $a_{\alpha i + 1}, a_{\alpha i + 2}, \ldots, a_{\alpha (i+1)}$
```

where $i = (b_0, b_1, \ldots, b_{dim-1})_2$ and constitutes the matrix : $D_i = \begin{bmatrix} a_{\alpha i+1} & 1\,0 \ldots 0 \\ a_{\alpha i+2} & 0\,1 \ldots 0 \\ \ldots & \ldots \\ a_{\alpha(i+1)} & 0\,0 \ldots 1 \end{bmatrix}$

```
PAR
    On each processor do
    SEQ
        Perform sequential algorithm on $D_i$. We obtain one row : $L = [d_i, l_{i_1}, \ldots, l_{i_\alpha}]$
        where $d_i = \gcd(a_{\alpha i+1}, a_{\alpha i+2}, \ldots, a_{\alpha(i+1)})$.
        for k := dim downto 1 do :
        $\alpha = 2\alpha$
        If $(2^k > \text{proc} \geq 2^{(k-1)})$ then
        Send row L to processor $(b_{dim-1}, \ldots, b_k, \ldots, b_1, b_0)_2$
        else $\{\text{proc} < 2^{(k-1)}\}$
        Let R be a $n/2^{(k-1)}+1$ dimension vector. Receive $R = [d_{i'}, 0, \ldots, 0, l_{i'_1}, \ldots, l_{i'_\alpha}]$
        where $[d_{i'}, l_{i'_1}, \ldots, l_{i'_\alpha}]$ is the row send by processor $(b_{dim-1}, \ldots, b_k, \ldots, b_1, b_0)_2$.
        Let $S = [d_i, l_{i_1}, \ldots, l_{i_\alpha}, 0, \ldots, 0]$ be a $(\frac{n}{2^{k-1}}+1)$ dimension vector (S completed with $n/2^k$ zeros).
        Perform transformation between S and R. If (R[1] = 0) then L := S else L:= R.
        endif
Outputs: A solution of the equation is obtained in processor 0.
```

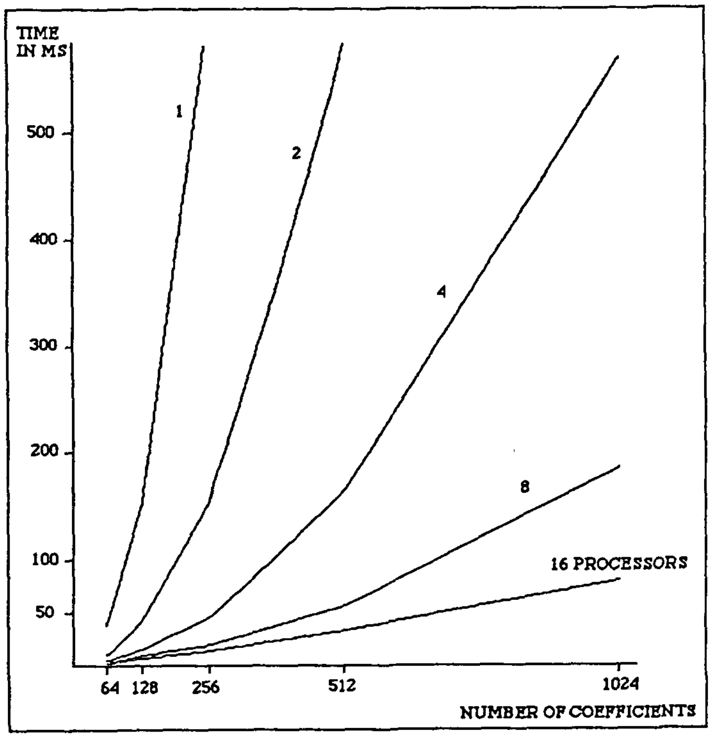


Figure III.1

Let us notice that, provided that the transformed rows S or R are preserved, we can construct the general solution. The non stored coefficients are all zeros.

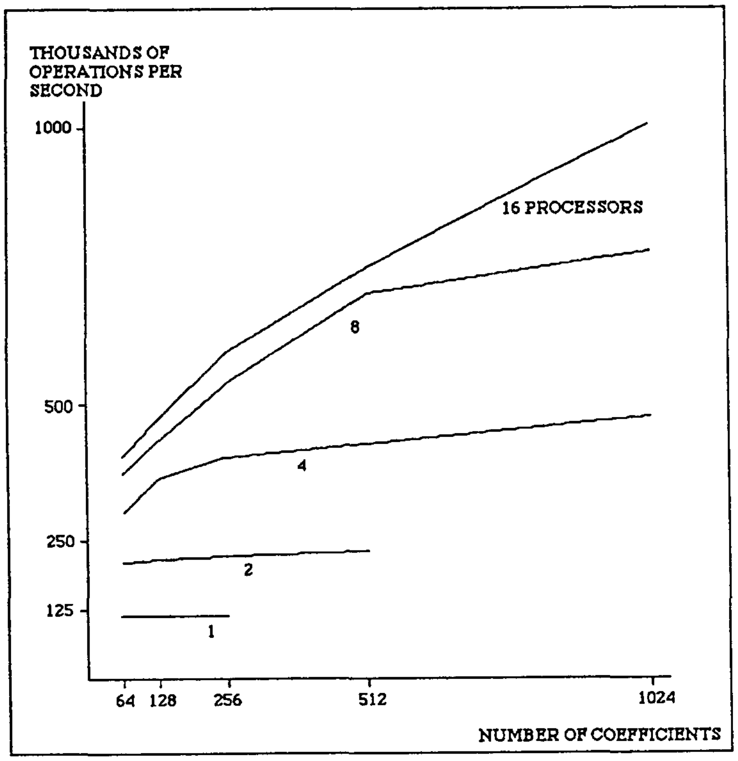


Figure III.2

On one node the tests have been performed for problems of size lower than 256 (we have to precise that the infinite–precision arithmetic was not yet available for those tests : the sizes of the coefficients was bounded by 2^{31}-1). The figure III.1 represents time in ms against number of coefficients and number of processors. Here the parallelism allows an important computation time saving.
The total number of manipulated data and the total number of performed operations decrease when the number of processors increases. These last remarks explain the time reductions obtained, for most cases a factor higher than two when the number of processors is only doubled.
The figure III.2 gives the performances obtained, in thousands of operations per second. For several processors the curves let appear communication time interferences.

III.2 / LINEAR SYSTEMS

III.2.1 / Gaussian elimination over GF(p) :

The numerical parallel algorithms for Gaussian elimination can easily be adapted to the GF(p) fields arithmetic. The main problem is to minimize the communication costs required when a null pivot is encountered. The detail of the implemented algorithms, the *Broadcast row* the *Pipeline ring* and the *Local pivot ring algorithms* can be found in [12] and a detailed presentation of the following results in [34]. Those algorithms can be implemented simply using the integer arithmetic of the Transputer, but the performances have been increased by developing a GF(p) arithmetic on the *Vector Processing Unit*.
The quantity usually considered to compare different parallel algorithms is the *efficiency* : the ratio of the

sequential execution time to the product of the parallel execution time by the number of processors. This definition not expresses that the number of operations does not depend on the number of processors. This hypothesis will not always be practically verified.

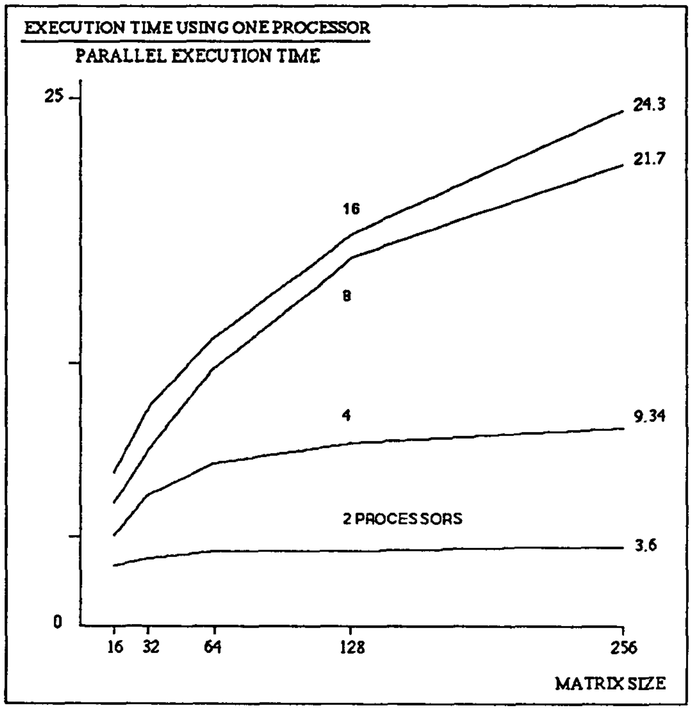


Figure III.3 : Local Pivots Algorithm,
Speed-up, modulo 7 calculus,
A(i,j) = 0 if j > i and 9097 otherwise.

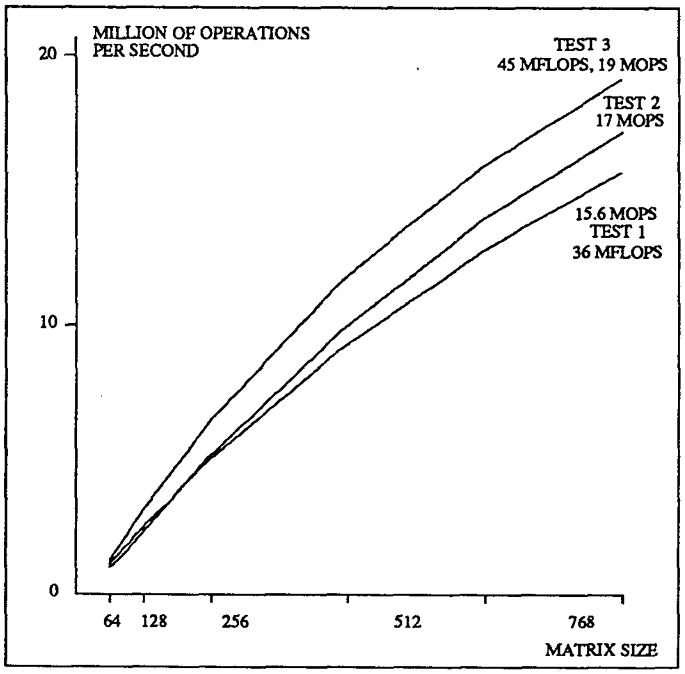


Figure III.4 : Performances, Millions of operations per second,
Test 1 : figure 1 matrix, pipeline ring algorithm,
Test 2 : figure 1 matrix, broadcast row algorithm,
Test 3 : random matrix (mod 22307), local pivot algorithm.

For a fixed size of matrix, the number of performed operations during the local pivots algorithm will depend on the entry matrix and on the number of processors (see [10] or [34]) : we show on figure III.3 below some measurements of speed–up : *the ratio is greater than the number of processors*. Figure III.4 shows us that 19 Mops (Millions of operations per second, +, * and mod) can be reached. Equivalently, 45 MFlops (floating point operations) are produced (the modulo needs five floating point operations).

II.2.2 / Linear systems over integers

A lot of algorithms provide the exact rational or integer solutions of a linear system. Solution can be obtained by a direct resolution; an alternative to reduce the *intermediary coefficients swell* is to use reductions modulo and p–adic expansions.

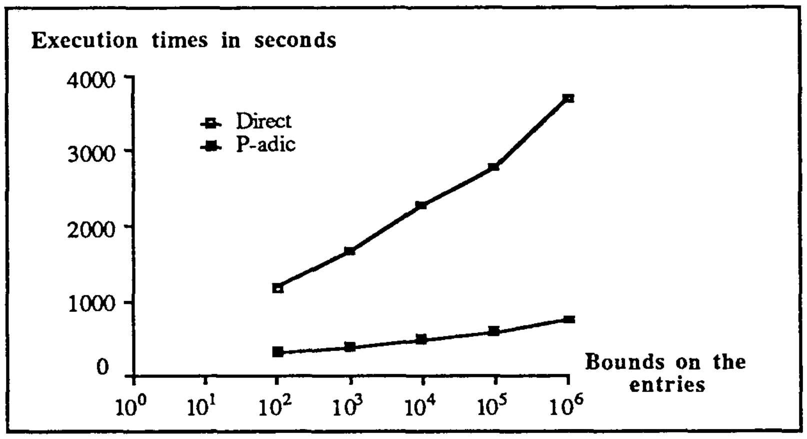


Figure III.5 : Execution times v.s. the sizes of the entries, for a 128*128 matrix (16 processors).

Two similar algorithms have been given in [Dix] and [GK]. These methods using p–adic expansions seem to be superior, in the case of large matrices (suitable to be treated on a powerful parallel machine), to methods using the Chinese Remainder theorem.

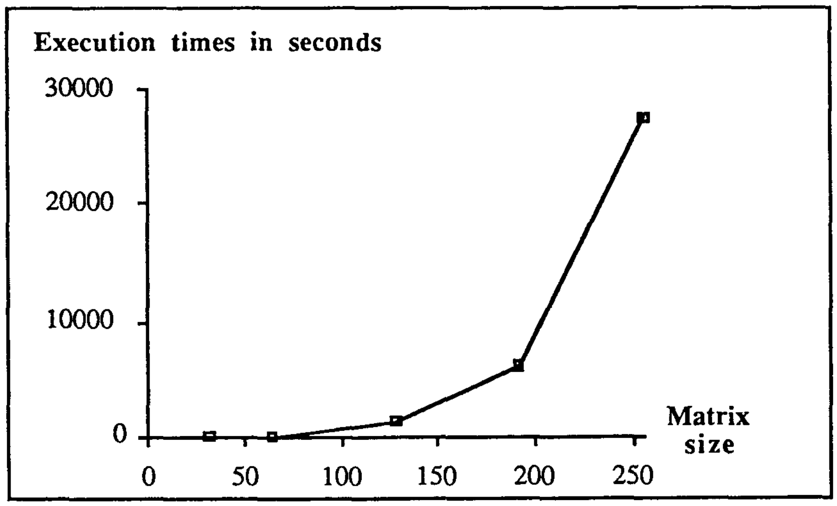


Figure III.6 : Executions times of the direct resolution, (entries bounded by 100).

We compare here, a direct implementation of the resolution (corresponding to the Bareiss' fraction free algorithm given in [Bar] and parallelized in [RSSV]) and the implementation of the p–adic resolution (which parallelization can be found in [Vi]). We have tested our parallel implementations to calculate the solutions of problems involving matrices with k–digit random elements, with values of k from 2 to 6. So within a factor at most 4/3 (in fact the one–step cost over the two–step cost [1]) we are in a context analogous to the one used for timing results given in [2, table III). As previously we can measure the efficiency of the algorithms.The hypothesis that the real cost of an algorithm is solely due to arithmetics and communications may no longer be verified in practice : an important extra–cost arises in algebraic computations, the variable–length arithmetic implies a memory management cost (which depends on the size of the available memory). The memory is more saturated during the sequential execution than the parallel one : this could lead to very surprising speed–up greater than the number of used processors.

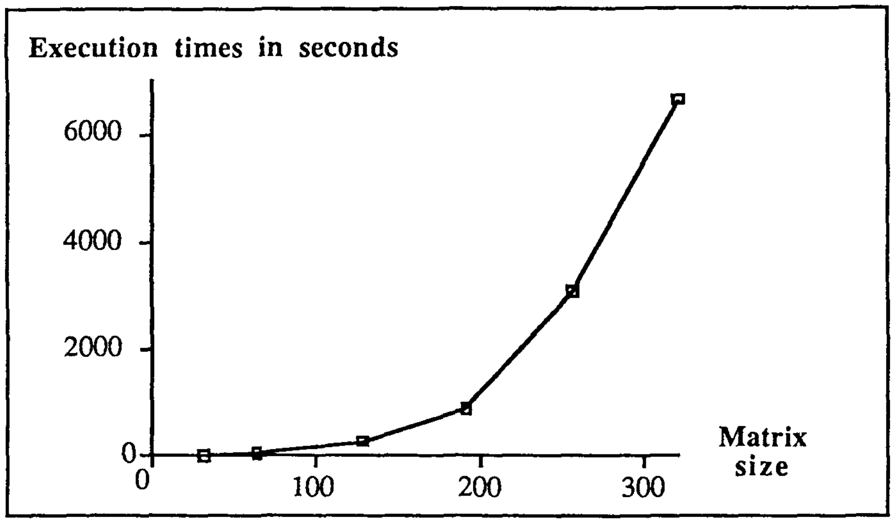


Figure III.7 : Execution times of the p-adic resolution (entries bounded by 100).

It is interseting to see on the figure III.5 the execution times versus the number of digits of the entries. Assuming we use a classical multiple–precision arithmetic, we recall here the theoretic sequential arithmetic costs : $O(n^5B^2)$ for Bareiss' algorithm [3], where B is a bound on the number of digits of the entries; and $O(n^3B\log^2 n)$ for the p–adic method. On the two last figures (III.6 and III.7) we present the execution times of the two resolutions. It appears as forecasted by the complexity studies, that the p–adic resolution is much better than the direct resolution.

IV / GRÖBNER BASES

IV.1 / General Presentation : the Parallel Algorithm

The definition and the way to compute a Gröbner basis is supposed to be known. None of the following notions will be developped: Critical pair, Spolynomial, normalisation. We want to parallelize the Buchberger 's algorithm [6]. A parallel algorithm is presented.We have determinated the independant tasks of the sequential algorithm and the proposed algorithm adds recursively adequate polynomials to the set of input polynomials. The fact that we work with boolean polynomials involves particular choices in the order of the variables, and simplifies the basic operations.

We work in $F_2 [x_1, \ldots\ldots, x_n] / _{(x_1^2+ x_1, \ldots\ldots, x_n^2+ x_n)}$ where F_2 is the field $\mathbb{Z}/2\mathbb{Z}$ and where $(x_1^2+ x_1, \ldots. x_n^2+ x_n)$ is the ideal generated by the polynomials $x_1^2+ x_1, \ldots x_n^2+ x_n$.

So we have the following properties:

i) $x_i^2 = x_i$ for all i in {1,..., n}.
ii) $x_i + x_i = 0$ for all i in {1,..., n}.

We assume to have at one's disposal a parallel machine with n processors, each having a local memory. The processors can be connected in order to form a ring. The algorithm presented below has been implemented on the hupercube FPS T20 of the TIM$_3$ laboratory.

IV.1.1 Computation of Spolynomials

The computation of the Spolynomials may be done in parallel since they are independant. If we have m polynomials in input and n (n is even) processors at one's disposal, with $m \geq n$., we distributate the m polynomials among the processors' memories in the following way:

$$m = qn + r \quad (0 \leq r < q)$$

(n - r) processors contain q polynomials
the r remaining processors contain q+1polynomials.

The main problem when computing the Spolynomials that is all the polynomials must meet each other.

i) In a first step, we compute in parallel, the Spolynomials associated to the polynomials contained in the memory of each processor. Let us suppose that k is the number of these polynomials
In a processor:
Beginning with FP = { $p_1, p_2, ...p_k$ } we compute C_{12} ,...., C_{1k}, C_{23}, ..., C_{2k}, ...,C_{k-1k} where C_{ij} is the Spolynomial associated to p_i and p_j.

ii)The subsets of input polynomials contained in each processor circulate along the ring which allows us to compute all the Spolynomial associated to the m input polynomials.

Let FP$_i$ be the collection of k polynomials contained in the ith processor of the ring. During the first step we have computed, in each processor, the Spolynomials associated to these polynomials. We shall note FC$_i$ the collection of Spolynomials obtained in the ith processor of the ring. We proceed as follow:

i) The collection FP$_i$ is transferred from the processor i to the processor i+1.
ii) The Spolynomials FC$_{i,i+1}$ from FP$_i$ and FP$_{i+1}$ are computed in each processor.
iii) FC$_i$ and FC$_{i,i+1}$ are concatenated in parallel .

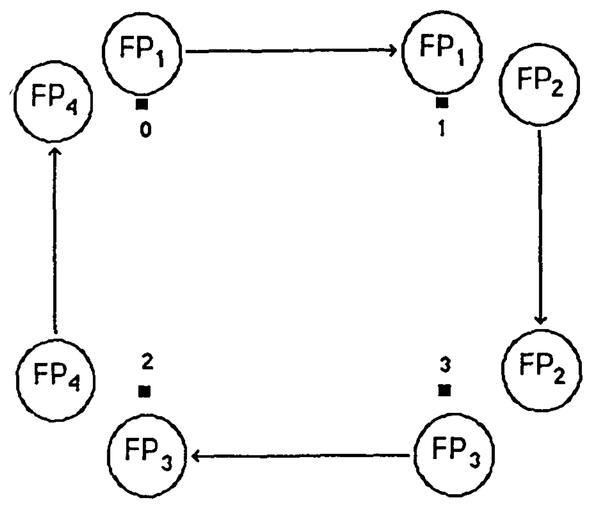
These three steps are then repeated until we have computed m (m-1)/2 Spolynomials associated to the m input polynomials. With four processors this manipulation may be represented in the following way :

First iteration:

ith processor of the ring ($i \neq 1$):

- FP$_i$ $\longrightarrow$FC$_i$
- send FP$_i$
- receive FPi-1
- (FPi, PPi-1) $\longrightarrow$ FCi,i-1

On a ring:



And so on, until having computed all the Spolynomials associated to the m input polynomials.

IV.1.2 / The normalization:

All the Spolynomials obtained must be normalized with respect to the input polynomials. The process is the same as during the computation of the Spolynomials. Each processor contains two collections of polynomials: the collection of its k input polynomials, and a collection of Spolynomials, resulting from the former steps.

The Spolynomials circulate along the ring and are normalized with respect to the given polynomials as soon as they meet them in the processors.
Then, the processors work in parallel to reduce the Spolynomials they receive with respect to the collection of the input polynomials they contain respectively.
The execution of this step is finished when all the Spolynomials are in normal form. The processors stop the computation simultaneously.

IV.1.3 / The whole algorithm:

When the Spolynomials normalized are computed, in order to obtain a Gröbner basis, the sequential algorithm repeats the same computation, substituting, at each iteration, the collection of input polynomials by the union of this collection with the collection of the Spolynomials normalized computed at the former step.Therefore, we may repeat, in the parallel algorithm, the step of computation and normalization of Spolynomials substituting, at each step and in each processor i of the ring, FP_i by the concatenation of FP_i and $FPCN_i$. We use this method, but modified, because it generates, at each step results yet obtained in the former step.

IV.2 / Implementation on the FPS T20

IV.2.1 / Choice of a representation for the boolean polynomials:

The choice of the structure to represent the boolean polynomials is justified by the fact that we want to translate the basic operations, such as the sum and the product of boolean polynomials, into a simple manpulation of that structure. This choice conditions the order on the variables. In a first time we have represented a boolean polynomial by an array of monomials, and each monomial by an integer. This integer is build (the integers are written in radix 2, with 32 bits) in the following way:

the monomial 1	is represented by 1
the monomial x_i	is represented by $2^i + 1$
the monomial $x_i x_j$	is represented by $2^i + 2^j + 1$

With this representation, and since the boolean operations are available on the integers, we easily translate the necessary operations for the computation of a Gröbner basis.
In order to represent polynomials with more than 32 variables, we now work using large numbers [28].

IV.2.2 / Results

The algorithm reads in input the number of polynomials, the number of processors wanted and the polynomials.
The following results show the evolution of the computing time of a Gröbner basis according to the number of processors.
The treatment of 32 polynomials with 5 variables gave us the following results:

num. of proc	time in sec
1	1.673
2	0.501
4	0.290
8	0.226
16	0.217

The time decreases as the number of polynomials increases. This decreasing is reduced by the delays of communications, which increase according to the number of processors . The more the computations are important in front of the communications the more the algorithm is interesting. It is difficult to control the number of simplifications required by the algorithm and then to know how the cost of communication grows according to the data. So we are studying an other algorithm where the communication cost does not depend on the simplifications. Instead of considering the independancy of certain tasks in the sequential algorithm, we use the following fact:
Let P= $(p_1, \ldots, p_k)$ a set of polynomials. Let $P_1 = (p_1, \ldots p_p)$, $P_2 = P - P_1$ and G_1 and G_2 gröbner bases associated respectively to P_1 and P_2.
A Gröbner basis of the union the G_1 and G_2 is also a Gröbner basis associated to P.
We do not describe the corresponding algorithm here.

REFERENCES

[1] H. Abelson, G. J. Sussman, J. Sussman "Structure & Interpretation of Computer Programs" (p. 491-503) Mc Graw-Hill Book Company (1985).

[2] A.V. Aho, J.E. Hopcroft, J.D. Ullman "Data Structure & Algorithm" (p. 378-407) Addison-Wesley (1983).

[3] E.H.Bareiss, "Computational Solution of Matrix Problems over an Integral Domain", J. Inst. Math. Applic. 10 (1972), 68–104.

[4] D. Bayer and M. Stillman "The Design of Macaulay: A System for Computing in Algebraïc Geometry and Commutative Algebra." (January 1986).

[5] W.A. Blankinship, A new version of the Euclidean algorithm, Amer. Math. Monthly, vol. 70, N°3 , (1967).

[61 B. Buchberger. "A Criticical Pair / Completion Algorithm for Finited Generated Ideals in Rings". Proc logic and Machines. Decision Problems and Complexity ed by E. Bröger, G. Hasenjaeger, D. Rödding.Springer LNCS 171 (1983).

[7] B. Buchberger. "Basic Features and Developpment of the Critical Pair / Completion Procedure". Preprint J. Kepler University Austria

[8] S.Cabay and T.P.L.Lam, "Congruence Techniques fot the Exact Solution of Integer Systems of Linear Equations", ACM Trans. Math. Software 3, 386–397 (1977).

[9] J. Chazarain. "The Lady, the tiger and the Gröbner Basis". Preprint n°100 University of Nice. Department of Mathematics.

[10] M.Cosnard and Y.Robert, "Implementing the Null Space Algorithm over GF(p) on a Ring of Processors", Second international symposium on Computer and Information Sciences, Istanbul (1987).

[11] M.Cosnard, B.Tourancheau, G.Villard, Présentation de l'hypercube T20 de FPS, Journées Architecture C3, Sophia Antipolis, Revue Bigre + Globule (1987).

[12] M.Cosnard, B.Tourancheau and G.Villard, "Gaussian Elimination on Message Passing Architectures", Proceedings of ICS 87, Athènes, Lect. Notes Comp. Sc. no 297, Springer Verlag (1988).

[13] J.D.Dixon, "Exact solution of Linear Equations using P–adic Expansions", Numer. Math. 40, 137–141 (1982).

[14] Floating Point Systems, "Programming the FPS T–Series, Release B, Portland Oregon 97223.

[15] G.A.Geist, "Efficient Parallel LU Factorization with Pivoting on a Hypercube Multiprocessor", ORNL Preprint 6211 (1985).

[16] G.H.Golub and C.F.Van Loan, "Matrix Computation", The John Hopkins Univ. Press (1983).

[17] R.T.Gregory and E.V.Krishnamurthy, "Methods and Applications of Error–Free Computations", Springer Verlag (1984).

[18] K.Hwang and F.Briggs, "Parallel Processing and Computer Architecture", Mc Graw Hill (1984).

[19] S.L.Johnsson and C.T.Ho, "Spanning Graphs for Optimum Broadcasting and Personalized Communication in Hypercubes", Technical Report 500, Comp. Sc. Dpt.,Yale University (1986).

[20] M.Kaminski, A.Paz, Computing the Hermite normal form on an integral matrix, Technical Report 417, Israel Institute of Technology (june 1987).

[21] D.E. Knuth "The Art of Computer Programming Vol. 2 : Semi-Numerical Algorithms" (p. 229-293) Addison-Wesley Reading Mass (1969).

[22] E.V.Krishnamurthy, T.M.Rao and K.Subramanian, "P–adic Arithmetic Procedures for Exact Matrix Computations", Proc. Indian Acad. Sci. 82A, 165–175 (1975).

[23] M.McClellan, "The Exact Solution of Systems of Linear Equations with Polynomials Coefficients", Journal of A.C.M., vol. 20, pp 563–588 (1973).

[24] D.G. Malm, A computer laboratory manual for number theory, student manual, COMPress (1980).

[25] R. Mœnck "Is a Linked List the Best Storage for an Algebra System" Research Report

[26] M.Newman, Integral matrices, Pure and applied mathematics, Academic Press (1973).

[27] E. Regener "Multiprecision Integer Division Examples Using Arbitrary Radix" ACM, vol. 10 N° 3 (1984).

[28] J.L.Roch, P.Sénéchaud, F.Siebert et G.Villard, "Parallel Algebraic Computing", Imag Grenoble, RR–686 I, (december 1987).

[29] J.L. Roch, P. Senechaud, F. Siebert, G. Villard "Calcul Formel, Parallelisme et Occam" OPPT Ed.T.Muntean (1987).

[30] Y. Saad, Topological properties of hypercubes, Research report YALEU / DCS / RR-389 (1985).

[31] Y.Saad, "Gaussian Elimination on Hypercubes", in Parallel Algorithms and Architectures, Eds. M.Cosnard & al., North-Holland (1986).

[32] A. Schrijver, Theory of linear and integer programming, John Wiley, Chichester, England (1985).

[33] Q.F.Stout and B.Wager, "Intensive Hypercube Communication : Prearranged Communication in Link–Bound Machines", CRL–TR–9–87, University of Michigan (1987).

[34] G.Villard, "Parallel General Solution of Rational Linear Systems using P–adic Expansions", Proceedings of the IFIP WG 10.3 Working Conference on Parallel Processing, Pisa Italy , Elsevier Sc.P. To appear (1988).

[35] S. Watt. "Bounded Parallelism in Computer Algebra". Thesis presented to the University of Waterloo Ontario (1985).