

Execution platform for high consuming parallel applications: a case study for Gröbner basis Position Paper

Jean-Charles Faugère

Bertil Folliot

Céline Boutros Saab

LIP6/CNRS Université Paris VI

case 168, 4 pl. Jussieu, F-75252 Paris Cedex 05

April 28, 2000

Abstract

Solving polynomial system is very common in various fields of mathematics, physics, and so on. The main tool for solving this problem is "Gröbner basis". Practical complexity is at most exponential both in time and space. This kind of applications have a crucial needs in computing resources.

We have developed at the University Paris 6 a parallel algorithm (F_4) for computing "Gröbner basis". We describe in this paper a first implementation of the F_4 algorithm. It runs on a set of heterogeneous workstations and needs a lot of manual administration in order to produce complete results. Manual administration includes: execution of a profiler in order to partition the data among the hosts, managing a list of available hosts (traditional placement tools are not well adapted to such complexity, specially from the memory point of view), managing faulty hosts or fault in the computation (with checkpoint and rollback to a previous computation step).

Considering this kind of applications runs for weeks and needs several gigabits, the manual administrator becomes rapidly the bottleneck of the computation. Based on our practical experience, we propose an execution platform dedicated to high consuming parallel applications. We show that part of the administration can be operated dynamically.

1 Introduction

We aim to build an execution platform for automatic management of scientific application with high resource consumption. These applications differs from classic ones by an important execution time and a large amount of memory used in an unpredictable way. As an exemple of a scientific application we consider the problem of solving polynomial equations systems with Computer Algebra tools. One of the main tools for solving algebraic systems is the computation of Gröbner bases (also called standard bases) (see [Buc65, Buc70, Buc79, Buc85, Dav93]). From a theoretical point of view the complexity in the worst case is very bad $O(2^{2^n})$ where n is the number of unknowns. In practice, however, the experimental cost of the algorithms is far better but extremely difficult to predict even for a human expert: huge systems of several Mega bytes can be solved in few minutes while small systems with 10 variables can be untractable with today computers/algorithms.

Moreover, the execution environment is composed of a set of networked workstation which are shared by many users and which may fail.

Stopping the execution and losing all the results already produced because of a fault, a machine's overload or a lack of memory may cause the application to never terminate. Existing solutions are usually based on checkpoints. This is insufficient for applications which fail because of memory shortage. In that case re-executing the application from the checkpoint will reproduce the same error. This problem is difficult to handle because the application itself should be changed in order to try a new configuration, which hopefully will use less memory. This problem may also occur when workstations are overloaded as a new distribution has to be found.

In this paper, we focus only on the placement problem for such kind of applications. In practice, the main objective of a placement algorithm is to improve the application's response time. This is done by placing the application's processes on a networked workstation. The decisions are usually taken according to coarse grain information (global memory, global amount of communications, ...). The placement decision, for efficiency reasons, relies on a restricted number of information. This technique is efficient for small processes (with relatively short duration, and consuming few resources), where a bad placement affects slightly the global execution. On the other hand, for such Gröbner basis application's processes which have an important execution time, traditional placement approaches are not appropriate. A bad placement of a such process can lead to an important augmentation of the response time. We propose to realize a fine observation of resources with a variable granularity depending on the needs of the placement algorithm.

2 Multi-criteria placement algorithms

A placement algorithm takes place at the processes' level, distributing them on the workstations in order to improve the application's response time. The usual placement criteria are the reduction of communications between stations, the distribution of memory usage and the load balancing [BSS93]. In the literature, there are two types of placement, dynamic and static. A static placement takes into account the application's structure and the system configuration without considering the resources usage. However, with a dynamic placement, as in GatoStar [FS94] or Utopia [ZZWD93], the resources states and their evolutions are taken into account and have an influence on the decisions. Involved criteria are the memory allocations, the system calls, the load of the stations and the I/Os. The parameter's values depend on the system evolution and a continuous observation is essential.

For instance GatoStar, developed at the University Paris 6, is a fault tolerant load sharing facility. It consists of a ring of hosts which exchange information about hosts' functioning and processes' execution. Having this kind of information online allows to automatically recover the system and quickly return it to operation, thus increasing the availability of the distributed resources. Each host maintains a load vector which contains a view of all host loads. Periodically, each host sends its vector to its immediate successor. Load messages are also used to detect host failures. Processes are recovered from a previous checkpoint and communication messages are "replayed" from a log. All this part is automatically managed by GatoStar [SF98]. The process allocation algorithm can be operated according to the following criteria: hosts load, process execution time, required memory, and communication between processes. In the application description the programmer can specify appropriate allocation criteria.

However, allocation criteria are on a process basis, and can not be switched dynamically during execution (for instance from memory intensive to computing intensive). More generally, the main limitation of most algorithms are that they are based on a coarse grain observation of resources (percentage of occupied memory, load of the workstation, ...) and dependencies between processes. This granularity does not reflect the exact state of the system because it supplies an approximate information and sometimes out of date (depending on the observation range). In addition, the information collected is often insufficient to build an automatic execution management of an application and to make a dynamic migration's algorithm, a checkpoint insertions and a new distribution.

3 Towards a fine grained observation

We propose an approach based on a more fine grained observation of the parameters. Such observation is expensive if it is systematically done, whereas a more finer grained is only necessary during the decision steps [BDB95]. The importance of a variable granularity approach is that it allows to reduce significantly the cost decreasing the amount of communications. Such approach try to reduce the perturbations while supplying the maximum of information. Thus, placement algorithm have the information at the time of the decision which is, in addition to the classical data, the communication volume, the parallelisation degree, the memory really used as well as a response time estimation.

The architecture of observation is distributed on a local network with a control server and a set of observation agents on each site. The server subscribes to a set of information. Only this information is sent back to the server. The agents save locally all the gathered information for a post-mortern analyses. The implementation lay on a Network Message Server (NMS) layer already developed in the LIP6 lab. The NMS provides an uniform and transparent control integration of tools management [FF94]. The tool in our case is the adaptive grain monitoring. Figure XX describes the prototype we are building. It is composed by "monitoring agent" running on each host on top of the NMS, and a "control agent" centralized, in charge of selecting dynamically the information granularity. When the monitored system is stable, the cost of monitoring is reduced by increasing the granularity of the information (time between information exchange and amount of data transmitted). When the system enter an unstable phase, the appropriate "monitoring agents" are selected by the control agent to decrease the information granularity. Only needed information is gathered and transmitted over the network, severely reducing the overhead of the monitoring. This information will then be used by our load sharing policy, that is still to developp.

4 Distributed observation architecture

The architecture of observation is distributed on a local network with a control server and a set of observation agents on each site. The server subscribes to a set of information. Only those informations are sent back to the server. The agents save locally all the gathered information for

a post-mortem analyses. The implementation lay on a NMS (Network Message Server) layer already developed in the LIP6 lab.

5 Sketch of the algorithm

The algorithm F_4 is designed to solve polynomial systems of equations but it relies heavily on linear computations. We first describe a parallel and efficient method for solving linear equations then we give a sketch of the sequential algorithm and a description of the parallel version.

5.1 How to solve linear systems of equations

Let b be the number of bits of the hardware arithmetic ($b = 32$ or 64).

We explain briefly how one can solve a linear system $Ax = b$ in parallel when A (resp. b) are $n \times n$ (resp. $n \times 1$) matrices with integer coefficients. In computer algebra integers have arbitrary lengths (the only limit is the size of memory), they are called bignums, as opposed to machine integers. In practice each entry of the matrices can have hundred or thousand of digits. One of the most efficient way to solve linear systems is to use multi-modular computations. Let p_1, \dots, p_h be small prime numbers (by small we mean $p_i < 2^b$). For each prime we can solve the subproblem $A_i x_i = b_i$ where $A_i = A \bmod p_i$ and $b_i = b \bmod p_i$. This computation is much faster than the original problem since all the computations can be done with the hardware arithmetic (the complexity is $O(n^3)$ where n is the size of the matrix A). From this point we can apply the Chinese Remainder Theorem and applying the formula:

$$x_{l:k} = \sum_{i=1}^h \left(\frac{p_{l:k}}{p_i} \right) \lambda_i x_i \text{ where } p_{l:k} = \prod_{i=1}^h p_i, \lambda_i = \frac{p_i}{p_{l:k}} \bmod p_i$$

and it is easy to see that we have constructed a solution of the problem $A_{l:k} x_{l:k} = b_{l:k}$ where $A_{l:k} = A \bmod p_{l:k}$ and $b_{l:k} = b \bmod p_{l:k}$. The solution x of the original problem is not a vector of integers but a vector of rationals $(\frac{u_1}{d_1}, \dots, \frac{u_n}{d_n})$; in order to recover the true solution we have to apply the following lemma:

Lemma 5.1 (*Lifting lemma*) *If $|u_i| < B$, $0 < d_i < B$ and $2B^2 \geq p_{l:k}$ then we can compute u_i and d_i from $(x_{l:k})_i$ by applying the extended gcd algorithm for the integers to $(x_{l:k})_i$ and $p_{l:k}$ ($(x_{l:k})_i$ is the i -th component of the vector $x_{l:k}$).*

In practice things are a little more difficult since we do not know B in advance (even it is possible to compute a rough estimate of B). The second difficulty is that sometimes (the probability is small but not zero) p_i divides $\det(A)$ and the computation $A_i x_i = b_i$ failed. In that case we say that p_i is a “bad prime”.

As a final remark we notice that the problem of reducing a matrix to a row echelon form is particular case of the previous problem.

5.2 Description of the F_4 algorithm

We use the notations of [Bec93] for basic definitions: k is the ground ring, $R[x] = k[x_1, \dots, x_n]$ is the polynomial ring. In practice k will be \mathbb{Z} the (big) integers or $\frac{\mathbb{Z}}{p\mathbb{Z}}$ the integers modulo a prime p . We denote by T , the set of all terms in these variables. We choose $<$ an admissible ordering on T . We give only a sketch of the algorithm F_4 and we refer to [Fau99] for a complete description:

Algorithm F_4

Input: $\begin{cases} F \text{ a list of polynomials } [f_1, \dots, f_m] \\ < \text{ an ordering of } T \end{cases}$
Output: a list of polynomials.
 $ToDo := \{Pair(f_i, f_j) \mid 1 \leq i < j \leq m\}$
 $r := 1$
while $ToDo \neq \emptyset$ **do**
 $Sel := Select(ToDo)$
 $ToDo := ToDo \setminus Sel$
 Construct the matrix $A_r := \text{Symbolic Preprocessing}(Sel, [f_1, \dots, f_m])$
 $M_r := \text{Reduction to Row Echelon Form of } A_r$
 $r := r + 1$
 for $h \in Rows(M_r)$ **such that** $h \neq 0$ **do**
 $m := m + 1$ **and** $f_m := h$
 $ToDo := ToDo \cup \{Pair(f_m, f_i) \mid i < m\}$
return $[f_1, \dots, f_m]$

Thus the algorithm can be viewed as a succession of matrix computations each computation can be decomposed in two steps: a “symbolic” step (this step generates the matrices A_1, A_2, \dots computed by Symbolic Preprocessing) which does not depend on the ground field k and the real computation (the matrices M_1, M_2, \dots computed by Reduction to Row Echelon Form). Actually the Symbolic Preprocessing function returns a list $[(u_{i_1}, f_{i_1}), \dots, (u_{i_k}, f_{i_k})]$ where $u_j \in T$ is a term and the constructed matrix is:

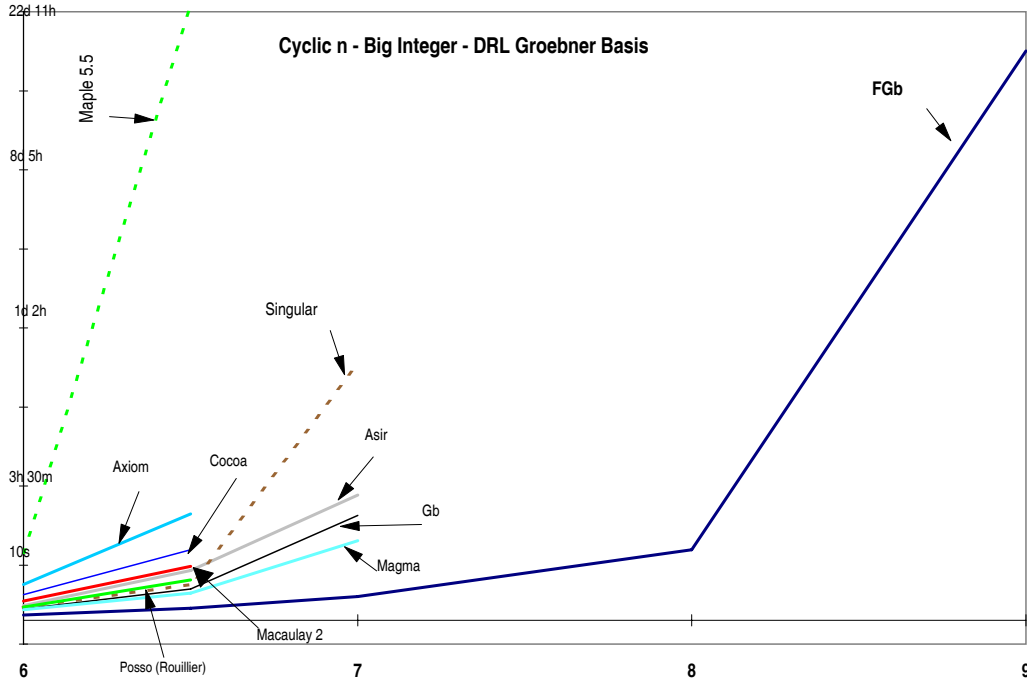
$$A = \begin{matrix} & t_1 & t_2 & t_3 & \dots \\ \begin{matrix} u_{i_1} f_{i_1} \\ u_{i_2} f_{i_2} \\ \vdots \\ u_{i_k} f_{i_k} \end{matrix} & \begin{pmatrix} \times & \times & 0 & \dots \\ \times & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \dots \\ 0 & \times & \times & \dots \end{pmatrix} \end{matrix} \quad (1)$$

where t_1, t_2, \dots is the set of all the terms occurring in $[u_{i_1} f_{i_1}, \dots, u_{i_k} f_{i_k}]$ and \times means a non zero element. Hence the coefficients of matrix are shared between the rows and we have to store only the “shape” of the matrix that is to say the indexes of the non zero elements. The shape of matrix is also independent of the ground field. For each row j of the matrix we associate the

number i_j such that the coefficients of f_{i_j} are the non zero entries of the row j (the order is the monomial ordering $<$). After the reduction to row echelon form the shape of the matrix is:

$$M = \begin{matrix} & t_1 & t_2 & & t_k & t_{k+1} & & t_m \\ \begin{matrix} f_{m+1} \\ f_{m+2} \\ \\ f_{m+r} \end{matrix} & \begin{pmatrix} 1 & 0 & \dots & 0 & \times & \dots & \times \\ 0 & 1 & \dots & 0 & \times & \dots & \times \\ & & \ddots & & \times & \dots & \times \\ 0 & 0 & \dots & 1 & \times & \dots & \times \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix} \end{matrix} \quad (2)$$

the sequential F_4 algorithm is much faster than other algorithms (mainly the Buchberger algorithm):



Comparison of the F_4 algorithm for a standard benchmark *Cyclien*

5.3 Description of the parallel algorithm

In the current version of the implementation the following tasks are managed by the operator:

a) We compute

$$\begin{aligned} p_0 &= \text{prevPrime}(2^b) \\ p_{i+1} &= \text{prevPrime}(p_i) \text{ for } i = 1, 2, \dots \end{aligned}$$

where $\text{prevPrime}(a)$ return the biggest prime p such that $p < a$.

b) we apply the F_4 algorithm mod p_0 and we store in a database D :

- the shape of the generated matrices A_1, A_2, \dots, A_d
- the time t_0 of the computation.
- the amount of memory B_0 use by the program.

c) let P_1, P_2, \dots, P_c the current list of available processors and B_i the physical memory of the processor P_i .

Parallel Algorithm F_4

Input: $\begin{cases} F \text{ a list of polynomials } [f_1, \dots, f_m] \\ D \text{ a distributed database} \end{cases}$

for $j := 1$ **to** d **do**

$l := 1, \text{todo} := \text{"compute"}$

failed:= 1

while $\text{todo} \neq \text{"end"}$ **do**

for $i := 1$ **to** c **do**

$\delta := \text{floor}(\frac{B_i}{B_0})$

$l_i := l, h_i := l + \delta, l := l + \delta + 1$

for $i := 1$ **to** c **do**

run on the processor P_i : Reduction to row echelon form A_j modulo $p_{l_i:h_i}$.

if the computation failed **then**

failed:= failed $\times p_{l_i:h_i}$

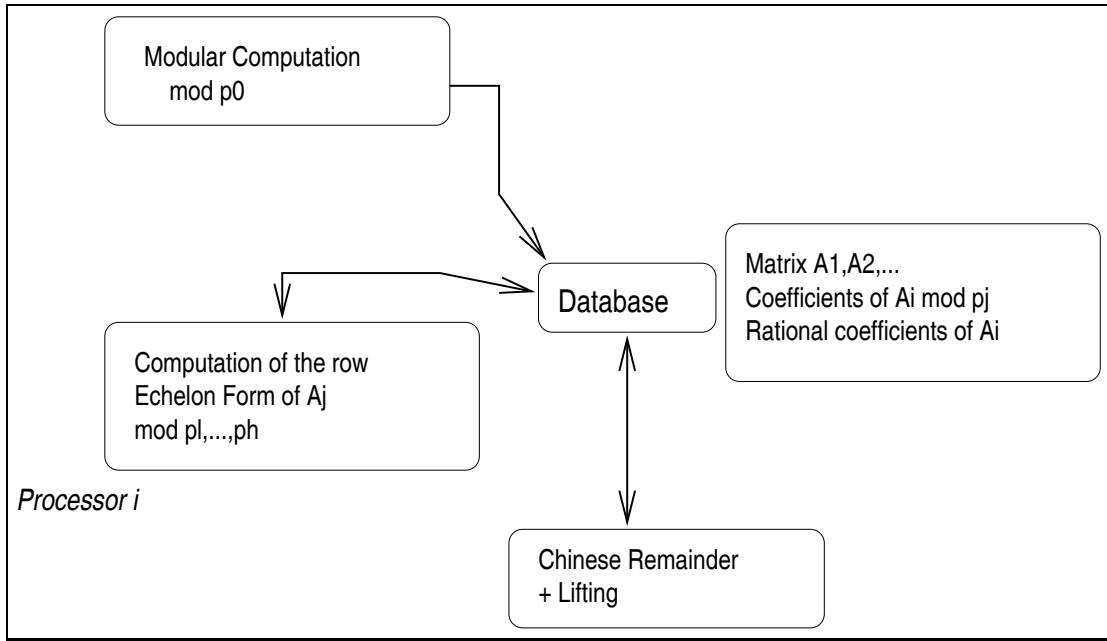
Compute the row echelon form A_j modulo $\frac{p_{1:l}}{\text{failed}}$ using the Chinese Remainder formula.

Try to recover the row echelon form of A_j in \mathbb{Q} using the lifting lemma.

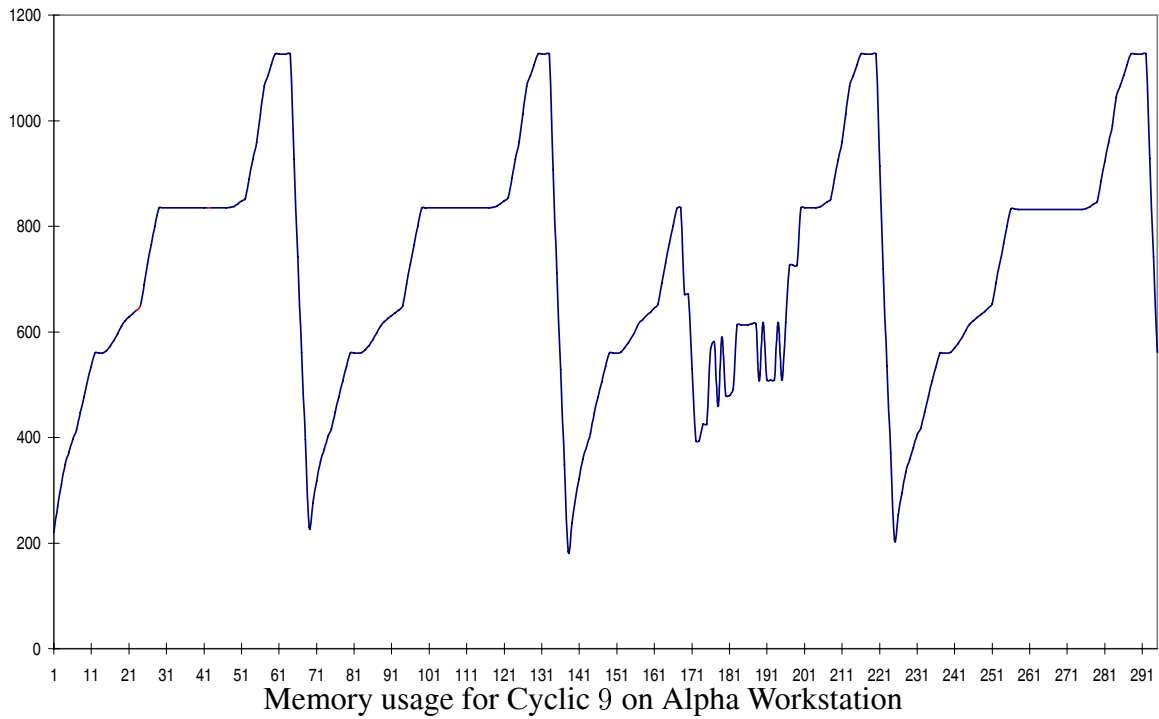
if the lifting lemma does not failed **then**

Store in the database the coefficients of A_j

$\text{todo} := \text{"end"}$



the estimated CPU is $\approx t_0 * \log(B)$ where T is the maximum coefficient in the result.



6 Some measures

The figure 1 represents the evolution of RSS (Resident Set Size) during the execution of the FGB application for two different polynomial system. One of them is a polynomial system with 7

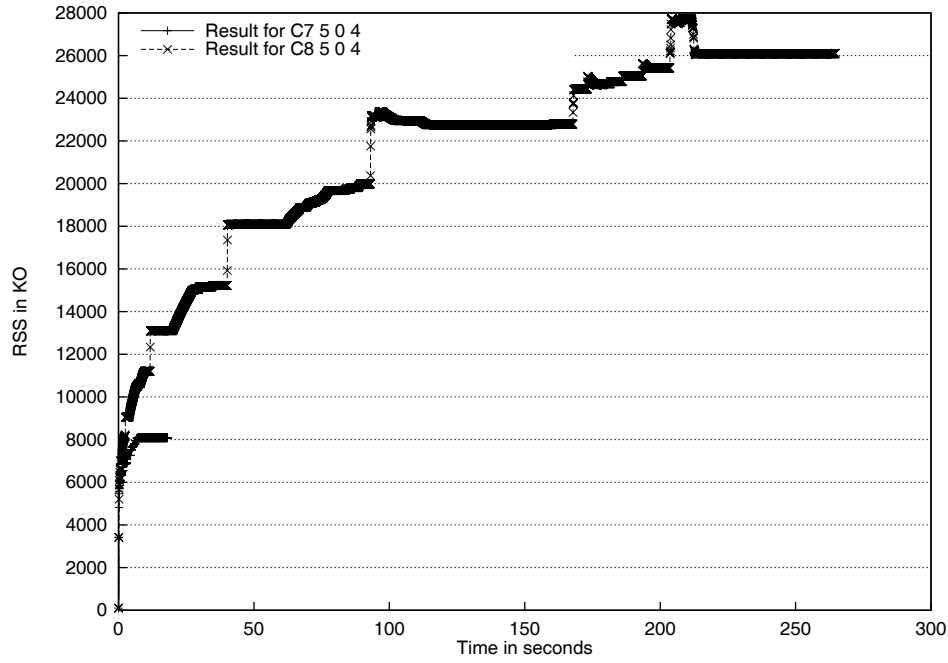


Figure 1:

variables and the second one has 8 variables. We notice that there is no relation between the two curves (we wanted to represent also the curve of the polynomial system of complexity 6 but it was invisible on the graph). In fact, the two curves have the same form but they are not related; from observing the execution time or the RSS allocation of the application for 7 variables, we can't deduce the RSS needed nor the execution time for an execution of an application with 8 variables. This is due to the irregularity of this application which is directly related to data input.

The figure 2 represents the evolution of RSS during the execution of the FGB application for polynomial system with 8 variables but for different number of vector. We notice that the general shape of the curves is the same; the allocation of RSS is proportionnal to the number of vector. However, the execution of a system with 8 vectors is faster and less RSS consuming then the same system with 7 vectors. This is certainly due to a memory shortage and is directly related to the memory and cache size of the workstation. We can consider that the system with 8 variables is the most suitable on this configuration, and it is interesting to find the relation between the memory and cache size and the execution time.

7 Conclusion

Traditional load sharing execution platform are adapted for a number of high consuming application. We have presented the case of the FGB application used for computing Gröbner basis. Based on our experience with several execution platform developments, GatoStar for load sharing and fault tolerance, and the Network Message Server for integration of control in soft-

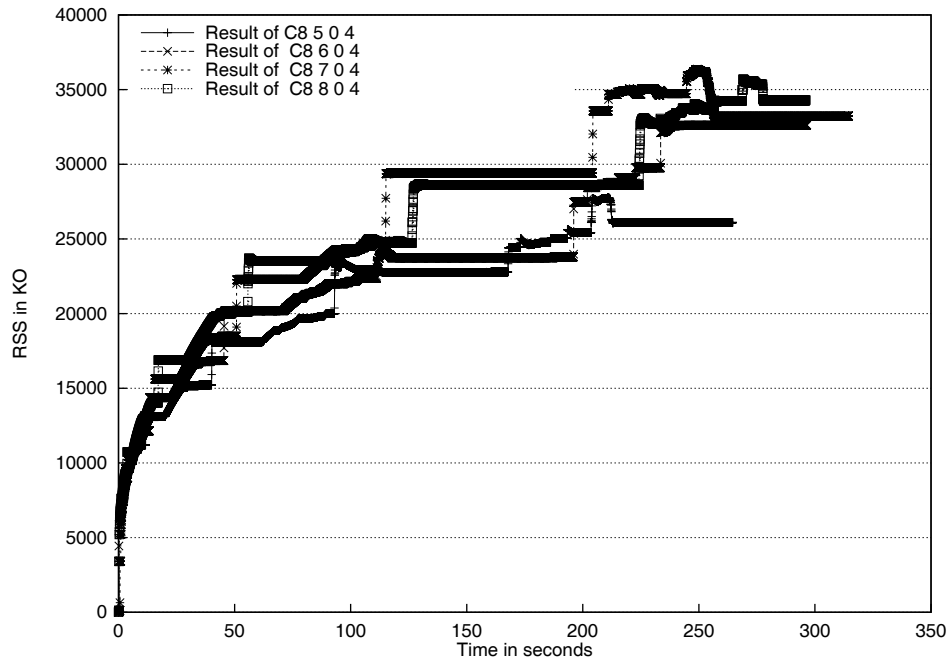


Figure 2:

ware environment, we proposed an other approach to execute applications such as FGb. We have shortly described the adaptative grain monitoring environment, that will be the basis of our new execution platform. The selected information will then be interpreted by our allocation algorithm in order to manage data placement, migration and execution rollback when needed resources are not available, as well as classical load sharing topics such as keeping track of available hosts and restoring the processes after a crash. We hope that such a platform will greatly simplify the task of the application designer and that execution administration can be mostly automatic. Moreover, several system tools (as graphical monitoring of the current state of the computation, or a distributed database keeping track of previous execution profiler), reduce both implementation and administration times.

References

- [Bec93] Becker T. and Weispfenning V. *Groebner Bases, a Computational Approach to Commutative Algebra*. Graduate Texts in Mathematics. Springer-Verlag, 1993.
- [Buc65] Buchberger B. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, Innsbruck, 1965.
- [Buc70] Buchberger B. An Algorithmical Criterion for the Solvability of Algebraic Systems. *Aequationes Mathematicae*, 4(3):374–383, 1970. (German).

- [Buc79] Buchberger B. A Criterion for Detecting Unnecessary Reductions in the Construction of Gröbner Basis. In *Proc. EUROSAM 79*, volume 72 of *Lect. Notes in Comp. Sci.*, pages 3–21. Springer Verlag, 1979.
- [Buc85] Buchberger B. Gröbner Bases : an Algorithmic Method in Polynomial Ideal Theory. In Reidel, editor, *Recent trends in multidimensional system theory*. Bose, 1985.
- [Dav93] Davenport J.H. and Siret Y. and Tournier E. *Calcul Formel*. Masson, 1993. 2^e édition révisée.
- [Fau99] Faugère J.C. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1–3):61–88, June 1999.
- [BDB95] A. Baggio, D. Prun, X. Bonnaire. Intrusion Free Monitoring: an observation engine for message server based applications. In *Proc. 10th Int. Symp. on Computer and Information Sciences*, pp. 541-548, 1995.
- [BSS93] G. Bernard, D. Stve, and M. Simatic. A Survey of Load Sharing Networks of Workstations. *Distributed Systems Engineering Journal*, 1, 1993.
- [FF94] Karim Foughali, Bertil Folliot. TOPIC-SE: Tool Based Open Platform for Integration of Control Software Environments. *Information and Software Technology*, 36(7):427-433, 1994.
- [FS94] Bertil Folliot, Pierre Sens. GatoStar: A Fault Tolerant Load Sharing Facility for Parallel Applications. *Dependable Computing EDCC-1*, Berlin, Germany. *Lecture Notes in Computer Science 852*, K. Echtle, D. Hammer and D. Powell eds. Springer-Verlag, pp. 581-598, 1994.
- [SF98] Pierre Sens, Bertil Folliot. The STAR Fault Tolerant Manager for Distributed Operating Environments. *Software — Practice and Experience*, 28(10):1079-1099, 1998.
- [ZZWD93] S.Zhou, X.Zheng, J.Wang, P. Delisle. Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software — Practice and Experience*, 23(12):1305-1336, 1993.