Parallel Calculation of Polynomial Roots according to Durand-Kerner Method using PVM

C. BALAYER, H. GUYENNET, F. SPIES
Laboratoire d'informatique
UFR des Sciences et des Techniques
25030 BESANCON Cedex, FRANCE
phone 33-81-66-64-57 fax 33-81-66-64-50
E-mail : balayer@univ-fcomte.fr

1 Introduction

The key obstacle to the widespread adoption of parallel systems is the difficulty in program development. An application has to be decomposed into a large number of parallel components, referred to as processes, objects according to the computational model underlying the programming language, and each component has to be assigned, or mapped, to the system node. Both program development and mapping are complex and a proper set of integrated programming tools is required, but it should not be designed for a physical machine. As in sequential systems, a virtual machine enables the development of applications that are portable across different hardware plateforms. PVM supplies the functions to automatically start-up tasks on the virtual machine, to manage communication and synchronisation among the processes, to enter and exit the virtual machine. PVM supports heterogeneity managing all data conversions, and distributes computation among loosely complex computation elements, ie among a set of network connected workstations.

2 The Durand-Kerner Method

We have chosen an asynchronous distributed application, with processes which not end at the same time and with the duration of life which is not known *a priori*. It is also necessary that the distribution of calculation time and communication time of a process would be as balanced as possible.

The Durand-Kerner method seems to be the better adapted to the problem we wish to study. This algorithm is iterative and converge by successive approximations towards the set of roots. The advantage of this method is to allow an asynchronous parallelization where each process of the application will be able to calculate a part of polynomial roots. These processes will require to exchange information during their running phase.

We want to find the roots ω_1,...,ω_n of a n-degree complex polynomial.

$$P(z) = \sum_{i=0}^{n} a_i z^{n-i} \ with \ a_0 = 1, \ a_n \neq 0 \ and \ a_i \ in \ C$$

The Durand-Kerner method is a simultaneous resolution method which allows the extraction of all the roots at once. It builds a sequence $Z^{(k)} = H(Z^{(k-1)})$ in C^n, from any initial vector $Z^{(0)}$ where H is an operator in C^n constructed in such a way that the $Z_i^{(k)}$ tends to the ω_i, $i = 1, ..., n$. In this method, the used operator is the Weierstrass operator :

$$H_i(Z) = z_i - \frac{P(z_i)}{\prod_{j=1, j\neq i}^{n} (z_i - z_j)} \ ; \ i = 1, ..., n.$$

The initialisation of the vector $Z^{(0)}$ is realised by the Guggenheimer method. This first vector $Z^{(0)}$ is made up of the set of roots. This initialisation is primordial, because it does not need that two elements of the vector converge to the same root. The Guggenheimer method consists in calculate a radius σ_0 from the coefficients of the polynomial, and like this the roots are placed at equidistance on the circle.

3 Adaptation of the Method for a High Degree

We modify the method because the *C language double type* is not sufficient to evaluate the function when a point is too far from a root.

We modify the Horner scheme to calculate immediatly the logarithm of the function $Q_n(z) = \ln(P(z))$.

We remind that :

$$\begin{aligned}\forall(a,b)\in C^{*^2};\ln(a+b) &= \ln(a)+\ln(1+\tfrac{b}{a})\\ &= \ln(a)+\ln(1+e^{\ln(b)-\ln(a)})\end{aligned}$$

Hence,

$Q_0(z)=\ln(a_0)$

and,

$\forall i\in[1,n]$;

$$\begin{cases} if\ a_i\neq 0\ Q_i(z)=Q_{i-1}(z)+\ln(z)\\ \qquad\qquad\qquad +\ln(1+e^{\ln(a_i)-Q_{i-1}(z)-\ln(z)})\\ else\ Q_i(z)=Q_{i-1}(z)+\ln(z)\end{cases}$$

4 Different Parallel Solutions

When we elaborate a parallel solution of Durand-Kerner method, we establish that the vector $Z^{(k)}$ is necessary as data parameter to determine $z_i^{(k+1)}$. We use two solutions to solve this problem.

4.1 Synchronous Algorithm

We calculate the overall $z_i^{(k+1)}$ to constitute $Z^{(k+1)}$. In this solution, the $z_i^{(k+1)}$ must be terminate before the beginning of the following step of the $z_i^{(k+2)}$. This involves a more or less important waiting phase at each end of iteration.

The synchronous algorithm is composed of two types of processes :

- the processes *calculation* which calculate a root of the polynomial using the Durand-Kerner method,
- the process *roots* which has to distribute data on the calculation processes and to collect the result of their calculation.

The calculation processes are organized in a unidirectional ring. This topology allows a data traffic (the roots) without using a broadcast. Each process receives only one root at every message, but when it needs it.

The algorithm is synchronous because the message exchanges take place during the calculation and at a given time. It is decomposed in steps which are the following for a polymonial of degree 4 :

- Step 0 : the process *roots* distributes data around the ring. Each calculation process owns a root that it will be calculated immediatly.
- Step 1 : the processes *calculation* send their root to their next process all around the ring. The received root allows an execution of a first product for Durand-Kerner.
- Step 2 : the processes *calculation* send the received root to their next process. A second product can be executed.
- Step 3 : the processes *calculation* send the received root to their next process. A third product can be executed. At this time each process can execute an estimation of the local root. If the stop condition is not reached, we go back to the step 0 else to the step n.
- ...
- Step n : each process *calculation* sends its result to the process *roots.*

In fact the ending is not so easy. The calculation of each of processes ends independently of each other according to the stop condition. When a process ends its calculation, it sends its result to the process *roots.* But it continues to execute in order to no cut the ring. When the process *roots* has received all the roots, it sends a token which will turn around the ring to indicate to the processes *calculation* that they can stop.

Fig. 1 illustrates the progress of this algorithm.

4.2 Asynchronous Algorithm

There is not really the concept of $Z^{(k)}$ because, for each improved root, the vector is updated and the $z_i^{(k+1)}$ can be calculated without waiting the result of the others processes. This solution avoids the inactived phases, but it needs more iterations to converge towards the wanted result.

The asynchronous algorithm is based on the *Farming* technique. A *Farmer* process prepares data then sends them to *Worker* slave processes which do the calculations and send back the result to the *Farmer.* The *Farmer* process attends to the initialisation of the polynomial and the solution vector. Once this initialisation realised, we calculate the solution sub-vector that each process will have to establish. These priliminary calculations being realised, we send all necessary data to each *Worker*, i.e :

- the solution sub-vector to calculate
- the polynomial
- the initial solution vector.

After this sending, the *Farmer* waits for the result of the *Worker.*

Polynomial Degree	PVM		
	1 station	2 stations	4 stations
10	43 iterations - 8sec	67 iterations - 10sec	
20	149 iterations - 31sec	159 iterations - 29sec	
50	547 iterations - 257sec	603 iterations - 200sec	850 iterations - 146sec
100		1660 iterations - 1450sec	2700 iterations - 1000sec

Table 1: Results obtained with the asynchronous model

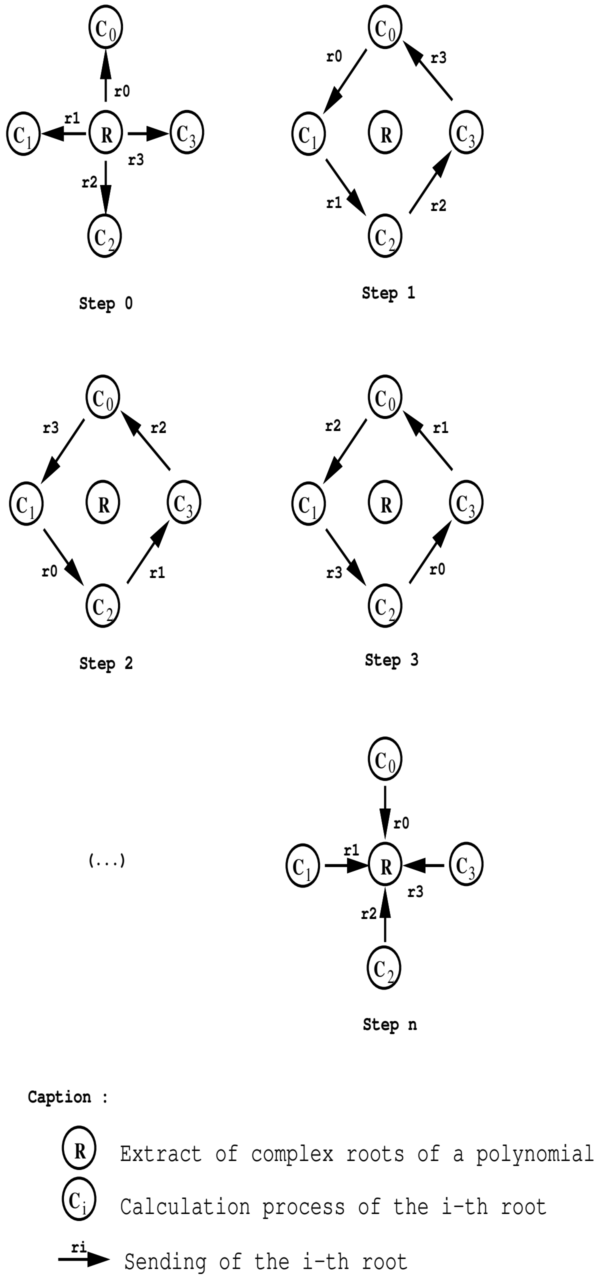

Figure 1: Synchronous model

The *Worker* processes calculate the roots. After data reception from the *Farmer*, they begin the calculation of their part of the solution vector; for each iteration, they send their part of the calculated solution vector to all others *Worker* using a broadcast message and in the same way they test if the others *Worker* have send to him data. In this case they update the solution vector. If there is not message from the others processors, they continue with the old values, from which one concludes the asynchronous characteristic of this algorithm. They send their solution once the stop condition satisfied.

The particularity of the *Worker* rests in the sending of the messages to the others *Worker*. We take advantage of the group notion in PVM. All the *Worker* are gathered in a same group. When a *Worker* process needs to communicate with the others, it does a broadcast of the message upon the group. PVM manages the sending to each process, but does not send the message to itself.

Fig. 2 show the progress of this algoritm.

5 Results

cf. Table 1

We can note that the asynchronous version has a tendency to generate more iterations than the synchronous one, since a node does not wait for the results of its neighboring.

In addition, the iteration number, for a given polynomial degree, increases with the station number, because the more there are stations the less they have to calculate roots. So if the station number is too important, there is a calculation over-cost because the information about the roots are too old.

6 Conclusion

We present an adaptation of the Durand-Kerner method, where we can extract roots from high degree polynomial.

The advantage brings by an executed environnement like PVM lies essentially in its rapidity of conception. The concept of parallel processes as well as the concept of message passing using several methods, like the point-to-point and the group, is directly

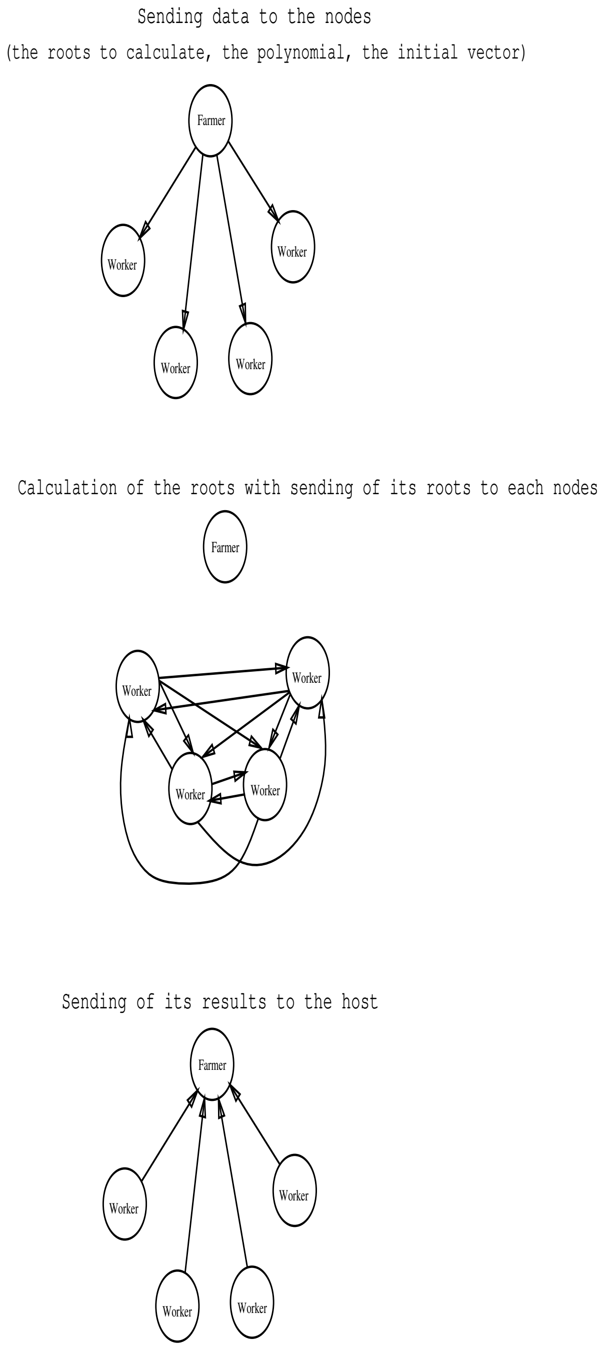

Figure 2: Asyncrhonous model

integrated. All these contributions accelerate the developpement of a distributed application.

The aim of our work consist in a generator of distributed applications which follows an exponential law. Thus, we will better appreciate the qualities of PVM to balance the load. PVM possesses a global view of the placement of its processes on the processors. However, the placement is made independently of the local load and the difference between the speed processors.

Bibliography

[**Cosnard 92**] M. Cosnard, D. Trystram
Algorithmes et architectures parallèles
Intereditions, 1992.

[**Cosnard 94**] M. Cosnard, P. Fraignaud
Analysis of Asynchronous Polynomial Root Finding Methods on a Distributed Memory Multicomputer
IEEE Trans. on Parallel and Distributed Systems, vol 5, N^o6, Jun 94.

[**Durand 60**] E. Durand
Solution numériques des équations algébriques
Masson, 1960.

[**Fraignaud 90**] P. Fraignaud
Communications intensives dans les architectures à mémoire distribuée
Ph. D. ENS Lyon, 1990.

[**Gerald 89**] C.F. Gerald, P.O. Wheatley
Applied Numerical Analysis
Addisson-Wesley Publishing Compagny, fourth edition, Juin 89.

[**Wilkinson 63**] J.H. Wilkinson
Rounding Errors in Algebraic Processes
Notes on Applied Science, N^o32, 1963.