PAC : First experiments on a 128 Transputers Méganode

Françoise ROCH-SIEBERT[1] and Gilles VILLARD[1]
Equipe Calcul Parallèle et Calcul Formel
LMC - CNRS 46, av. Félix Viallet, F-38031 Grenoble Cédex

Résumé. Lancé il y a trois ans, le projet PAC : *Parallel Algebraic Computing* [17], s'était donné pour but dans une première phase, de valider certains algorithmes de Calcul Formel sur un hypercube à 16 processeurs et de justifier l'utilisation du parallélisme. Dans son cadre nous continuons non seulement à généraliser les résultats qui avaient été obtenus, mais nous envisageons aussi de nouveaux problèmes. Le parallélisme maintenant utilisé est beaucoup plus massif, puisque les expériences sont menées sur un réseau de 128 Transputers. Nous présentons dans ce papier les premiers résultats expérimentaux obtenus. Ils concernent, à titre d'exemple, l'application du théorème chinois des restes en algèbre linéaire. Nous montrons comment, pour un nombre fixé de processeurs, la topologie choisie pour le réseau influe sur le comportement d'un algorithme. Nous insistons sur les coûts des communications et sur les contraintes liées à l'espace mémoire disponible.

Abstract. From its beginning three years ago, the PAC project : *Parallel Algebraic Computing* [17], has been exploiting a 16 processors hypercube to validate some Algebraic Computation algorithms, and to justify the use of parallelism. Going further, we begin to generalize the previous results and study new problems. Experiments are now hold on a more massively parallel computer : a 128 Transputers network. In this paper, we present the first results we have obtained : as an example, we have been interested in applying the chinese remainder theorem in linear algebra. For a fixed number of processors, we show how the behaviour of an algorithm is influenced by the chosen network topology. We point out the communication costs and the constraints due to the storage requirements.

1. Introduction

In the field of Computer Algebra, Takahasi and Ishibashi [20] developped in 1961 a new class of methods based on congruences, as an alternative to infinite precision arithmetic. Solutions are first obtained in Galois Fields(p) for various prime numbers p, and then recovered using an application of the Chinese Remainder Theorem [11].

A famous example of such an application is the exact solution of linear equations with integer, rational or polynomial entries [1, 2, 3, 4, 13], or matrix inversion. After a brief recall of the different algorithms which have been developped and theoretically discussed in [22], we describe the new architecture we are working on, and present the experimentations.

Precisely, we focus on the problem of computing n integers $(x_1, x_2, \ldots, x_n)$ from their residues modulo h prime numbers $p_1, p_2, \ldots, p_h$. We call $(x_{1k}, x_{2k}, \ldots, x_{hk})$ the residues of x_k $1 \le k \le n$. And we assume that $|x_k| \le B$ for $1 \le k \le n$ and $p \le p_i \le q$ for $1 \le i \le h$. Without any particular interest to the computation of the residues, we assume A(n) to be the sequential cost for each modulo p_i resolution (to compute the x_{ik}, $1 \le k \le n$), and $\varphi(n)$ the required memory. A(n) and $\varphi(n)$ count words of size $\log_2 q$, on such a word an arithmetic operation takes one unit of time.

For the parallel model we consider a network of P processors (numbered from 0 to P-1). We let $P = \alpha n$ and $P = \beta h$ where $0 < \alpha, \beta \le 1$. Each processor has a private memory and communicates by a message passing protocol (rendez-vous) with its neighbours [9]. The machine operates in an asynchronous MIMD mode [5]. The time to transfer n $\log_2 q$ sized words between two adjacent processors is modelized in the literature by $\beta + n\tau$ [18], where β is the communication start-up and τ the elementary transfer time. For our asymptotic analysis, we assume $\beta = 0$ without loss of generality [18].
With such a model, let

$$\boxed{\frac{A(n)}{P} + C_r(P)\tau} \qquad (1)$$

be the parallel cost on P processors for each modulo p_i resolution. Where, depending on the network topology, $C_r(P)$ is the communication cost. Thus $C_r(1) = 0$. The target topology for the whole algorithm is a two dimensional grid with wrap around, but we will just need (§2.2) a ring of processors for the modulo p_i resolutions.

[1]This work was supported in part by the *PRC Mathématiques et informatique* of the french *Centre National de la Recherche Scientifique* (CNRS).

And let

$$\boxed{\frac{\varphi(n)}{P}} \qquad (2)$$

be the required memory size on each processor.

Assumptions (1) and (2) of load-balanced executions are not too restrictive, see further details in [16, 17, 21].

2. The parallel algorithm

2.1 Basic recalls on chinese remaindering

The Chinese Remainder theorem is used to compute integers from their residues. We describe here the well known mixed-radix conversion algorithm given by a constructive proof of a consequence of the theorem [11 chap. 4.3.2, 12, 23]. Let $p_1, p_2, \ldots, p_h$, be h prime numbers, and $X_1, X_2, \ldots, X_h$, h integers. If P_t is defined as $P_t = p_1 p_2 \ldots p_h$, we can determine a unique integer X such that :

$$\frac{-P_t}{2} < X < \frac{P_t}{2}, \text{ and } X \equiv X_i \bmod p_i \text{ for } 1 \le i \le h.$$

The proof gives X using h steps of computation from its residues : working with the balanced representation of the elements m_i of the $GF(p_i)$, i.e. $-p_i/2 < m_i < p_i/2$ ($p_i > 2$) we define,

$$\begin{aligned} Y_1 &= X_1 \bmod p_1, \\ Y_2 &= (X_2 - Y_1)c_{12} \bmod p_2, \\ Y_3 &= ((X_3 - Y_1)c_{13} - Y_2)c_{23} \bmod p_3, \qquad (3) \\ &\ldots\ldots \\ Y_h &= (\ldots((X_h - Y_1)c_{1h} - Y_2)c_{2h} - \ldots - Y_{h-1})c_{h-1,h} \bmod p_h, \end{aligned}$$

Where the c_{ij} constants are given by :

$$c_{ij}\, p_i \equiv 1 \bmod p_j \text{ for } 1 \le i < j \le h.$$

The value of X is then computed as,

$$X = Y_1 + Y_2 p_1 + Y_3 p_1 p_2 + \ldots + Y_h p_1 p_2 \ldots p_{h-1}. \qquad (4)$$

The h-tuple $(p_1, p_2, \ldots, p_h)$ is called the h-tuple of radices associated to X. More generally, for each integer x_k of our problem ($1 \le k \le n$), we obtain the y_{ik} ($1 \le i \le h$) from (3) and the decomposition (4) becomes,

$$x_k = y_{1k} + y_{2k} p_1 + \ldots + y_{hk} p_1 p_2 \ldots p_{h-1}. \qquad (5)$$

To determine the quantity h, let us recall that the absolute values of the integers we have to recover are bounded by B. To place us under the conditions of the theorem, we have therefore to choose $P_t \ge 2B$. And since the prime numbers have all been chosen greater than p, h just has to satisfy :

$$h \ge \log_p B + \log_p 2,$$

we will simply use :

$$\boxed{h = \log_p B.} \qquad (6)$$

The predominant terms of the arithmetic costs of steps (3) and (4) are respectively $3h^2/2$ and h^2. Which gives for the n integers,

$$\boxed{\frac{5}{2} nh^2 + O(h^2)}. \qquad (7)$$

While the required memory size evaluated from (5) for the n integers, is asymptotically,

$$\boxed{nh + O(h)}. \qquad (8)$$

2.2 The parallel algorithm

● **A first approach :** a simple look at the whole congruence algorithm gives us a first parallel implementation. Indeed, the h modulo p_i resolutions are independent and can be done simultaneously : h/P resolutions on each processor. At the end of this stage each processor stores h/P residues for every x_k. The recovering stage have therefore to be performed in parallel. For a sequential implementation the same h-uple of radices is associated to all the x_k, there is no need to change the decomposition (5) from a x_k to another. Contrary to that, to avoid any global scheme of communication, we assume as in [22] that the target integers do not share a same h-uple of radices. We define allocp(i) for $1 \le i \le h$ as the number of the processor which stores the residues mod p_i. In the same way the application alloc(k) for $1 \le k \le n$ determines an equidistribution of the x_k over the network : processor $P_{alloc(k)}$ will store x_k at the end of the process.

On a ring of processors, for a processor P_{proc}, communications will simply consist in sending a message to P_{proc+1} (regarding the numbering of the ring) or receiving a message from P_{proc-1}. Let us give the basic idea we use [22] :

- Step 0 : every processor P_{proc} begins (3) using the h/P primes p_i such that allocp(i)=proc and for the n/P x_k verifying alloc(k)=proc.
- Step i : it sends his results (gathered with those received at step i-1) to P_{proc+1}. And goes on computing (3) : still using the same primes but now for the n/P x_k which partial decompositions have been received from P_{proc-1}.
- Last step : (5) can be computed without any communication.

We give now the algorithm for the processor P_{proc} ($0 \le proc \le P-1$), the communications are implicitly *from my predecessor* or *to my successor*. And let us consider a new numbering

of the primes : let $p_{1,proc}, \ldots, p_{h/P,proc}$ be the p_i such that allocp(i)=proc.

Algorithm DhT (Distinct h-Tuples)

```
for i = 0 to P-1
   { communications }
   PAR if i>0
      send (y1k, y2k, ..., yhi/P,k) for the k
            such that alloc(k)= proc-i+1 mod P
      receive (y1k, y2k, ..., yhi/P,k) for the k
            such that alloc(k)= proc-i mod P

   { computations of (3) }
   for j= 1 to h/P
      for k = 1 to n s.t. alloc(k)=proc-i mod P
         compute yhi/P + j,k modulo pj,proc
         according to (3)
for k = 1 to n s.t. alloc(k)=proc+1 mod P
   compute (5).
```

Observing that arithmetic operations are equidistributed, the arithmetic constituent of the cost is simply obtained from A(n) (the cost of each modulo resolution) and from (7) :

$$\frac{h}{P}A(n) + \frac{5nh^2}{2P} + O(h^2). \qquad (9)$$

Concerning the communications : each step i, $0 \le i < P$, asks for the transfer of hin/P^2 residues. With (9) we can so deduce the cost of DhT algorithm :

$$\frac{hA(n)}{P} + \frac{5nh^2}{2P} + \frac{nh}{2}\tau + O(h^2). \qquad (DhT)$$

The memory required on each processor is firstly the memory needed for one sequential modulo p_i resolution. In addition we have the equidistributed storage of the integers for the Chinese remaindering. Using quantity (8) over P we get at last :

$$\varphi(n) + \frac{nh}{P} + O(h). \qquad (MDhT)$$

The inherent parallelism (!) of h independent modulo p_i resolutions has been fully exploited. But the main drawback of such a choice will clearly appear during experiments (§4) : $\varphi(n)$ can be a serious restriction and limits the dimension n of the problem to solve.

● **A second approach** consists in exploiting parallelism before the Chinese Remaindering. According to (1), each modulo p_i resolution can be parallelized in such a way that the storage is distributed over the network (2). Without loss of generality we may assume that each processor computes all the residues for n/P among the n integers x_k . Obviously steps (3) and (5) are then sequential on every processor.

Algorithm SCR (Sequential Chinese Remainder)

```
{ if alloc(k)=proc, xk mod pi has been
computed for all i on this processor }

{ without communication }
for k = 1 to n such that alloc(k)=proc
   compute (3) and (5).
```

From (1) and (7) the corresponding cost is :

$$\frac{hA(n)}{P} + \frac{5nh^2}{2P} + hC_r(P)\tau + O(h^2). \qquad (SCR)$$

The storage is now fully distributed (2), on each processor we need :

$$\frac{\varphi(n) + nh}{P} + O(h). \qquad (MSCR)$$

Comparing (SCR) with (DhT) asks for the value of $C_r(P)$. This will be done in §4 considering applications of chinese remainders. But $C_r(P)$ could be a prohibitive cost for a use of (SCR) even if memory is saved up.

● Looking for a compromise between those two points of view, both stages of modulo resolutions and of recovering can be done in parallel if grids are viewed as cartesian products. The cartesian product of two graphs G=(X,E) and G'=(X',E') is defined as the graph F which set of vertices is X x X', and where the edge between (x,y) and (x',y') exists if and only if : x=y and $\{x',y'\} \in E'$ or x'=y' and $\{x,y\} \in E$.

lemma 1 : assume that P=RxS. A P processors 2-dimensional grid contains S distinct rings. The processors of those rings may be numbered from 0 to R-1 in such a way that, for $0 \le i < R$, the sets of processors i are R distinct S processors rings.

From this lemma we will use a double numbering of the processors. Assuming the current topology is a cartesian product with P= RxS :

$$\{ P_{proc} , 0 \le proc \le P-1 \}$$
$$=$$
$$\{ P_{r,s} , 0 \le r \le R-1 \text{ and } 0 \le s \le S-1 \}.$$

We also define two applications

allocp : [0, h] -> [0, S-1]

and

alloc : [0, n] -> [0, R-1]

for an equidistribution of the primes on S=P/R processors, and an equidistribution of the x_k on R processors. Algorithm ParCR [22] computes the modulo resolutions on the R processors topologies, and then uses S processors to recover each x_k.

Algorithm ParCR (Parallel Chinese Remainder)

- Each modulo p_i resolution is parallelized using the R processors $P_{r,proc}$ verifying allocp(i)=proc. Therefore, at a given time, S resolutions are simultaneously computed.

- At the end of this stage and for any k, the residues of x_k are distributed over the S processors $P_{pos,s}$ with alloc(k)=pos.

- Algorithm DhT is applied R times in parallel using S processors. Indeed, for each r, $0 \le r < R$, the S processors $P_{r,s}$ compute the integers x_k such that alloc(k)=r.

Figure 1 and figure 2 illustrate ParCR algorithm on a 24 processors grid, with R=6 and S=4. It is viewed as a cartesian product of two rings. The modulo p_i resolutions are simultaneously performed on each of the four "horizontal rings". Each "vertical ring" is then dedicated to four of the x_k.

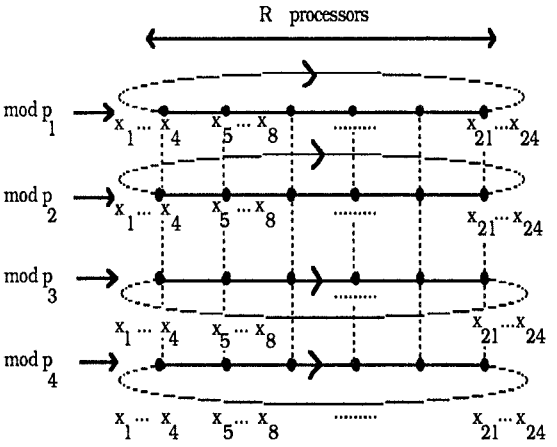


Figure 1 : ParCR algorithm, modulo p_i resolutions.

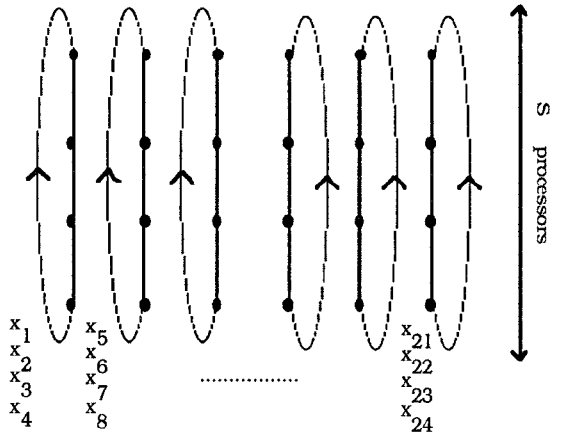


Figure 2 : ParCR algorithm, chinese remaindering.

2.3 Total costs

The modulo p_i resolutions : according to (1), each resolution can be parallelized with a communication cost $C_r(R)$ on R processors. During this first stage, S modulo resolutions are simultaneously executed : the cost is the one of h/S resolutions each using R processors. The total cost is then easily obtained from (1) and (2) :

$$\frac{h}{S}\frac{A(n)}{R} + \frac{h}{S} C_r(R)\,\tau,$$

or,

$$\frac{hA(n)}{P} + \frac{hR}{P} C_r(R)\,\tau. \qquad (14)$$

And the required memory space is,

$$\frac{\varphi(n)}{R}. \qquad (15)$$

Chinese remaindering : in the same way, the second stage consists in R simultaneous mixed-radix conversions each using S processors. The cost is the one of the recovering of n/R integers on a S processors ring. As for (DhT) we have :

$$\frac{n}{R}\left(\frac{5h^2}{2S} + \gamma(R)\frac{h}{2}\tau\right) + O(h^2),$$

or,

$$\frac{5nh^2}{2P} + \gamma(R)\frac{nh}{2R}\tau + O(h^2). \qquad (16)$$

with $\gamma(R)=0$ if R=P (S=1) and $\gamma(R)=1$ otherwise. The storage is equidistributed. Gathering together (14) and (16) we derive the total cost using a ParCR type conversion algorithm on a P processors grid. R is an integer lower than P which divides P exactly. Let $\gamma(R)$ be equal to 0 if R=P and to 1 otherwise. We have,

$$\boxed{\frac{hA(n)}{P} + \frac{5nh^2}{2P} + \frac{hR}{P} C_r(R)\,\tau + \gamma(R)\frac{nh}{2R}\tau + O(h^2).} \qquad (\text{ParCR})$$

The required space is $\varphi(n)/R$ during the first stage, data are then equidistributed. We get :

$$\boxed{\frac{\varphi(n)}{R} + \frac{nh}{P} + O(h).} \qquad (\text{MParCR})$$

We may now conclude in a theoretical point of view. Minimizing the total cost of the computation will therefore consist in finding the best compromise (best value for R) between the communication cost of modulo p_i resolutions and the one of the Chinese remaindering. But R=P is clearly the best choice if minimizing the storage is the main goal.

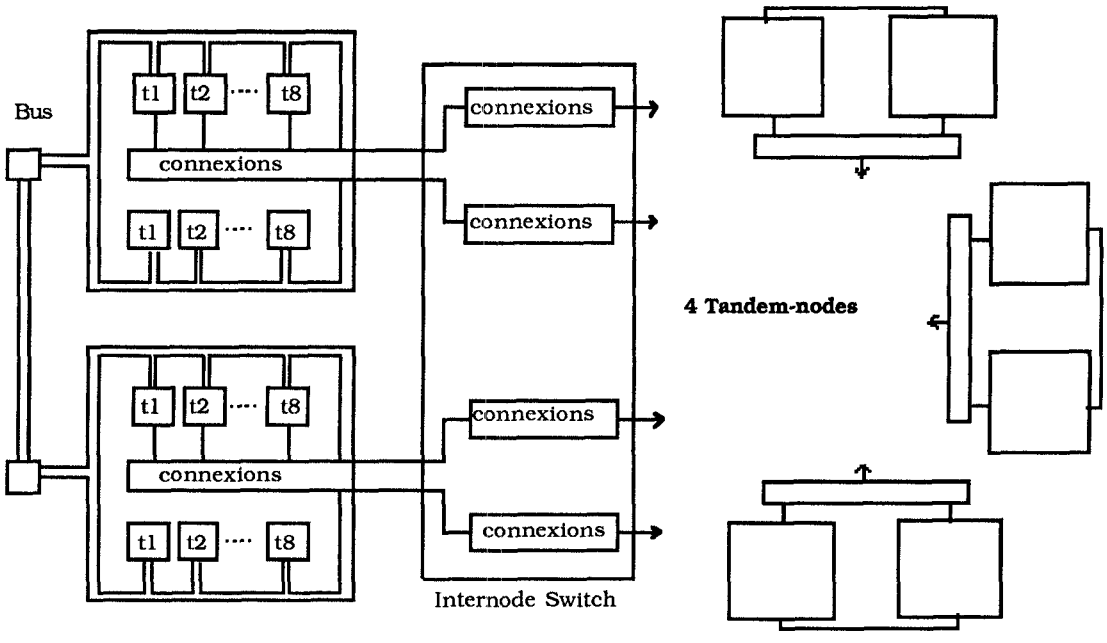


Figure 3 : the 128 transputers Méganode.

3. The 128 Transputers Méganode and its constraints

The Telmat Méganode [19] is a distributed memory multiprocessor computer which operates in an asynchronous MIMD mode : this exactly corresponds to our theoretical model. The Méganode may be considered as an experimental computer. The software environment is poor : codes are written in a C language augmented with local communication routines (between neighbours). Therefore, the main difference with our previous experiences [17] does concern nor the software nor the hardware but comes from the number of processors available : 128 versus 16.

The méganode is partitionned into 4 basic modules called tandem-nodes of 32 transputers (1 transputer T800 [10]+ 1Mb memory) each as shown on figure 3. On the inside of those tandem-nodes, a 72x72 crossbar allows to realize any degree 4 topology for the network (provide that no input/output is required, see below). A second level of connections (four 32x32 crossbars), the internode switch, then manages the communications between distinct tandem-nodes. Communications between transputers are done by message passing. We have the communication start-up :

$$\beta \approx 5{,}6.10^{-6}\text{s}$$

and ,

$$1{,}1.10^{-6} \leq \tau \leq 2{,}2.10^{-6}\text{s/byte}$$

for the throughput of the channels [14]. Any graph of degree 4 is avalaible, except the closed ones. Indeed, even if each transputer has four links, one link of the network is reserved for the data transfers to the host, especially for booting and for the input/output. Consequently a 128 processors 2d-torus for instance, cannot be realized. But smaller closed networks can easily be simulated using extra processors for routing the data. Figure 4 shows us the RxS grids we have been using for running ParCR algorithm (§2.2).

A column of R routing processors has been necessary during chinese remaindering : with a communication cost at most doubled. Dimensions had thus to verify $R(S+1) \leq 128$.

NB : the dynamic reconfiguration of links, that is the possibility to modify the network topology during the course of an execution, is an important tool for the conception of the parallel algorithms. Alas ! Even if it was an expected quality of the Méganode, it is still not at our disposal.

4. Applications, experimental results

As we mentioned in the introduction, a famous application of Chinese remaindering is the exact solution of linear systems [1, 2, 3, 4, 13]. It is also known that the method is effective while dealing with high dimensions of matrices Applying the results of previous sections, we discuss in this paragraph about implementations of Jordan eliminations for

matrix inversion and of Gaussian eliminations for linear system solution. Let A be a square matrix. Its determinant, say d, is non zero.

4.1 Modulo resolutions

We first report some measurements concerning linear systems Ax=b solution in GaloisField(p)'s, when p is a prime number less than $\sqrt{2^{52}-1}$ (prime numbers are coded on 64 bits). A general study of the problem can be found in [21]. Following this work, we have chosen to implement a pipeline ring algorithm as the fundamental brick of more general system solutions. Let n be the dimension of A. The first observation we make concerns the time to solve a fixed-size problem versus the number of processors. It is now well known that this has little sense using a distributed memory computer [6, 7]. Indeed, the performances of a 120 processors network will be correctly evaluated if a large system is solved : a larger system than those a sole processor could treat.

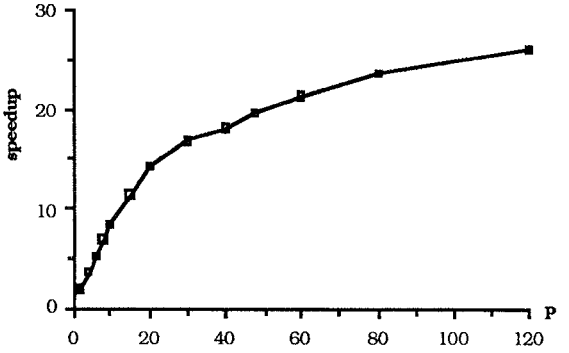


Figure 5 : speedup measurements for a 240*240 matrix (B=1000).

Therefore, the low speedups (ratios of the time spent by a single processor over the time spent by P processors) of figure 5 are not surprising if we have a look at figure 6 at the same time.

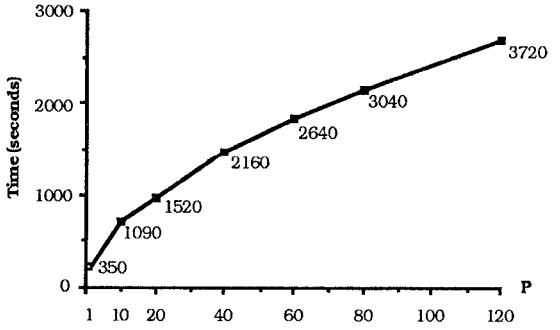


Figure 6 : greatest dimensions for problems Ax=b mod p that can be solved on P processors, and the corresponding costs (B=100).

The 120 processors not even allow to reduce the cost in a factor 30 just because the considered problem dimension is not representative of their capabilities. Another way to compute speedups which has been proposed by Gustafson [6, 7] will be used in the next paragraph.

Measuring the number of operations computed per second per processor, figure 7 points out the communications cost : it takes a more important part of the whole process while the number of processor increases.

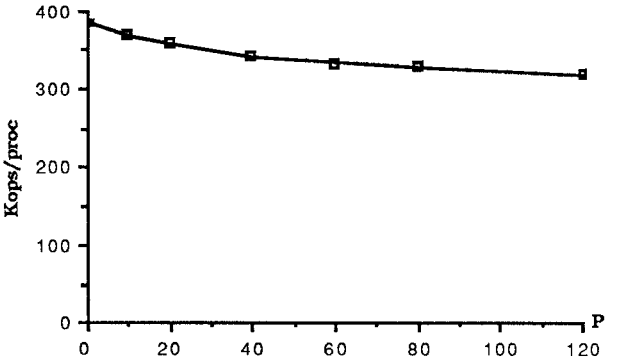


Figure 7 : thousands of operations per second per processor solving the greatest system on P processors.

4.2 Linear system solution

We now consider the linear systems Ax=b solution over the integers. A is still a n x n matrix. Its coefficients and those of vector b are absolutely bounded by B. We still assume the determinant d of A is non zero.

• The first stage of ParCR algorithm consists in performing h (given by Hadamard's inequality on |d| and (6)) modulo p_i resolutions. We assume the prime numbers p_i are great enough to neglect the pivoting cost. From [15] we may asymptotically consider :

$$C_r(R) \approx n^2 + Rn. \qquad (19)$$

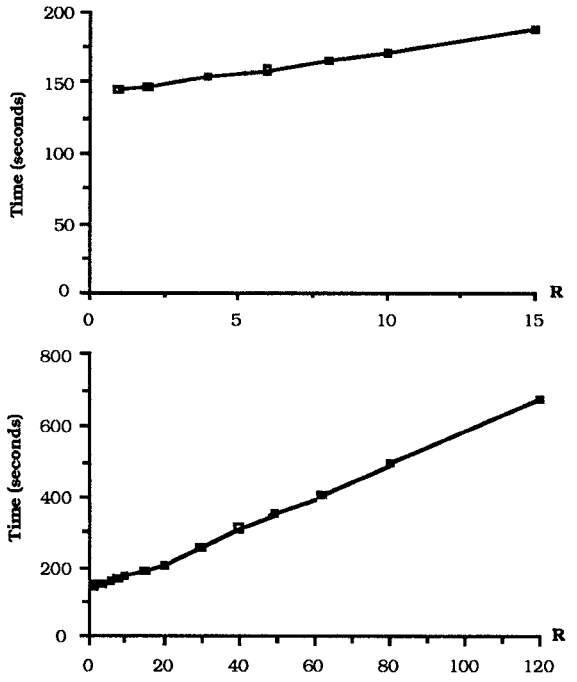
• Assuming that d is computed separately : the second stage reduces to apply the Chinese remaindering for the computation of the n numerators of the rational coefficients of vector x.

From (ParCR) we obtain the corresponding communication cost on a P=R x S processors grid (recall that $\gamma(x)=0$ if x=P and 1 otherwise) :

$$G_C(n,P,R) \approx h\left[\gamma\left(\frac{P}{R}\right).\frac{R}{P}(n^2+Rn)+\gamma(R).\left(\frac{n}{2R}\right)\right]\tau. \qquad (20)$$

For any given number P of processors and any value R≠1, this quantity is clearly dominated by the n^2 term. Its minimization is therefore

obvious : R=1 is the best choice. Compared with the low communication cost of the Chinese remaindering itself, any split of the modulo p_i resolutions over the network would always induce an extra-cost. This is well illustrated on figures 8 and 9. Considering R=120 costs five times as much as using R=1.



Figures 8 & 9 : linear systems solution, computational costs versus the value of R (B=100).

Real life problems ask to deal with matrices of high dimensions. At the time, a few percentage of gain is no more a priority if it seriously limits the dimension of the problem. Let us see what is happening for linear systems solution. A modulo p_i elimination needs at least the storage of $\varphi(n)=n^2$ coefficients. If M_{max} is the size of the workspace on each processor, from (MParCR), n must satisfy :

$$\frac{n^2}{R}+\frac{nh}{P} \le M_{max}. \qquad (21)$$

Assuming the absolute values of the coefficients of A and b are bounded by B the Hadamard's inequality gives us a bound for the integers x_k to recover :

$$B^n n^{n/2}.$$

Using (6) we may therefore asymptotically consider,

$$h \approx n\log_p nB. \qquad (22)$$

And condition (21) on n becomes :

$$\frac{n^2}{R}+\frac{n^2\log_p nB}{P} \le M_{max}. \qquad (23)$$

In other words, for any given dimension n of a system to solve, the value of R which minimizes the computational cost, is the lowest divisor R of P satisfying :

$$\boxed{R \ge \frac{n^2}{M_{max} - \frac{n^2\log_p nb}{P}}.} \qquad (24)$$

For large enough values of the lower bound p on the prime numbers p_i, n^2/R is the largest term of the memory cost. The maximum problem dimension n_{max} we may consider on P processors, then varies within a factor $\sqrt{P}$. We have roughly $n_{max} \approx \sqrt{M_{max}}$ if R=1 and $n_{max} \approx \sqrt{P}\sqrt{M_{max}}$ if R=P. To increase R is therefore a good way to exceed the limits imposed to the dimensions. On the Méganode 800 Ko available on each processor for the storage of the coefficients lead to $n_{max} \approx 300$ with R=1 (B=100) and to $n_{max} \approx 3000$ with R=120 (B=100). As pointed before, the gain of about 25% on the total computational cost from R=15 to R=1 for instance, (figure 8) could be hidden by the limits imposed to dimensions : 1100 versus 300.

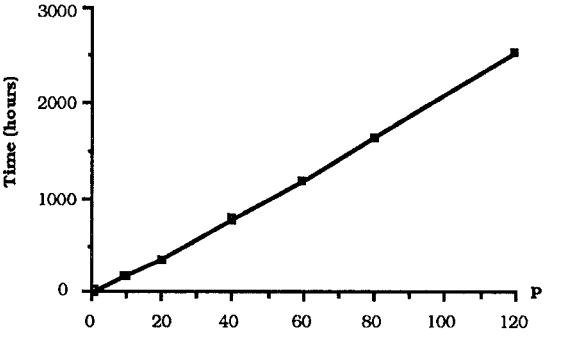
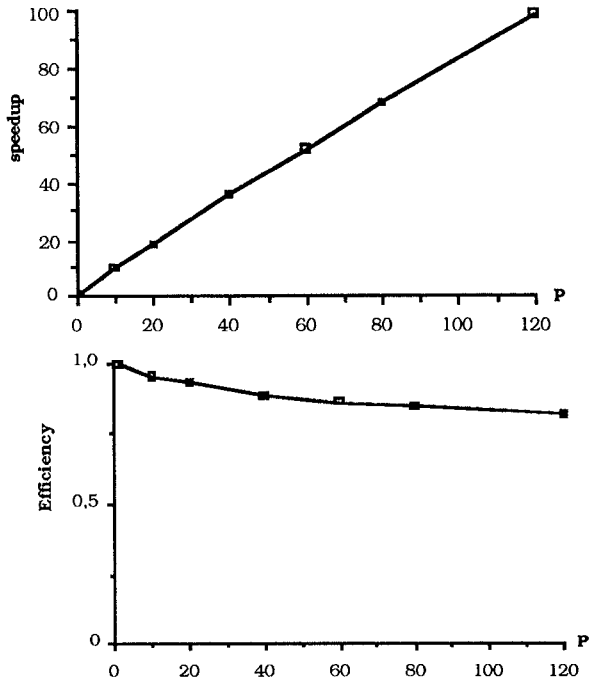


Figure 10 : greastest problems Ax=b (B=100) on P processors, extrapolated execution times.

We noticed in previous paragraph that computing speedups running a fixed-size problem is of low interest. Gustafson introduced [6, 7] a new way to obtain them. For each number P of processor, we consider the largest instance of the problem that can be solved and compute the time it would have taken on a single processor. The ratio between those times is the Gustafson's speedup. But the corresponding dimensions are too large in our situation : they not allow us to measure the required execution times (it would take about 2500 hours to solve a 3000x3000 system). We just present some strict extrapolations obtained from the execution times modulo p (figure 6).



Figures 11 & 12 : extrapolated Gustafson's speedup and efficiency (B=100).

The execution times are reported on figure 10. Speedups and efficiency are given on figures 11&12. The 120 processors grid reduces the cost of the resolution in a factor 100.

4.3 Matrix inversion

Let A now be a $\sqrt{n} \times \sqrt{n}$ matrix of integers bounded by B. Its determinant is non zero. The dimension of the problem is still n the number of rationals to compute : the coefficients of the inverse matrix A^{-1}. This inverse can be computed performing Jordan eliminations on A augmented with the identity matrix :

- We first perform h modulo p_i eliminations on A. Under the same assumptions than §4.2 we get from [21, 15] :

$$C_r(R) \approx n + R\sqrt{n}. \qquad (25)$$

- We also assume that d is computed separately : the second stage then reduces to apply the Chinese remaindering for the computation of the n cofactors of A from their h residues.

Applying algorithm ParCR, the communication cost of the whole process, is directly derived from (ParCR) :

$$J_c(n,P,R) \approx h\,[\,\gamma(\frac{P}{R}).\frac{R}{P}(n + R\sqrt{n}) + \gamma(R).(\frac{n}{2R})\,]\,\tau. \qquad (26)$$

Looking for the value of R which minimizes this quantity leads to complex formulas in the general case. The term n/2R dominates for the lowest values of R, while $(R/P)(n + R\sqrt{n})$ increases with R. Consequences are better illustrated by figures 13&14.

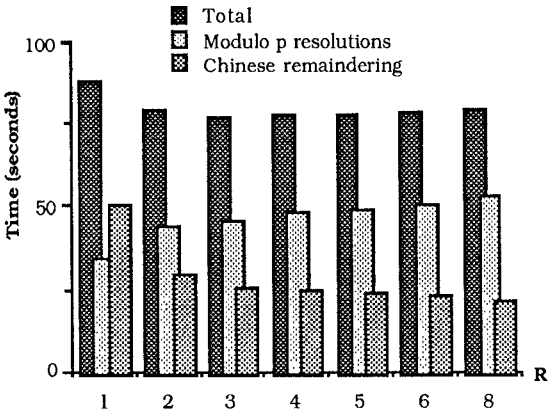


Figure 13 : total computation times for A^{-1} and repartition between the different phases versus the value of R (n = P = 120, B = 10^6).

For the inversion on 120 processors, the best value is R=3. The best compromise is reached when : each modulo resolution uses a 3 processors ring, while each recovering process uses a 40 processors ring. When P=80 our example leads to a best value R=5.

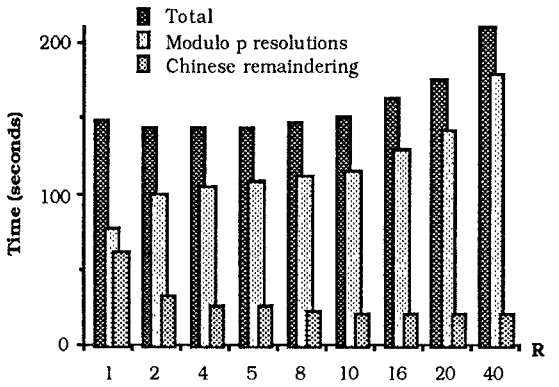


Figure 14 : total computation times for A^{-1} and repartition between the different phases versus the value of R (n = 160, P = 80, B = 10^3).

Let us observe before concluding, the respective contributions of the different phases in the total execution times. The time spent during the chinese remaindering process decreases with R while the time spent during the first step increases. And according to formulas the contribution of the chinese remaindering increases faster with B than with n (it is predominant in a unique case, R=1 and P=n=120).

5. Conclusion

Applying the Chinese remainder theorem in parallel to compute n integers, a main question arises : is it necessary to introduce a communication cost during the first phase ? Which number R of processors do we have to dedicate to each modulo resolution ? Let us recall that such a question may seems paradoxical since the work of this first phase is to solve h independent problems.

• If the goal is to minimize the total computational cost, our answer depends on the communication complexity of the modulo resolutions. If it is of an order O(n) : we have to find the value R which leads to the lowest total cost given by the quantity (ParCR). An illustration has been given by the problem of matrix inversion. On the contrary, if it is of an order greater than O(n), it dominates the whole communication cost. Therefore there is no need to parallelize the modulo resolutions, this would always induce an extra-cost : linear system solution is a representative example.

• Obviously, when the goal is to increase the dimensions of the solved problems, the solution is also to split the modulo resolutions. At the time, to minimize the total communication cost, according to the available memory we choose the lowest acceptable (see (24)) value of R.

References

[1] E.H. Bareiss, Computational solutions of matrix problems over an integral domain, *J. Inst. Math. Applic.* 10 (1972) 68-104.

[2] I. Borosh and A.S. Fraenkel, Exact solutions of linear equations with rational coefficients by congruence techniques, *Mathematics of Computation* 20 (1966) 107-112.

[3] S.Cabay, Exact solution of linear equations, *Proc. Second Symp. on Symbolic and Algebraic Manipulation*, ACM, New-York (1971) 392-398.

[4] S. Cabay and T.P.L. Lam, Congruence techniques for the exact solution of integer systems of linear equations, *ACM Trans. Math. Software* 3 (4) (1977) 386-397.

[5] M.J. Flynn, Very high-speed computing systems, *Proc. IEEE* 54 (1966) 1901-1909.

[6] J.L. Gustafson, Reevaluating Amdahl's law, *Communications of the ACM* 31 5 (1988) 532-533.

[7] J.L. Gustafson, The scaled-sized model : a revision of Amdahl's law, in *I C S Supercomputing 88*, L.P. Kartashev and S.I. Kartashev eds., International Supercomputing Institute Inc. (1988), vol. II, 130-133.

[8] J.L. Gustafson, S. Hawkinson and K. Scott, The architecture of a homogeneous vector supercomputer, *Proc. ICCP 86*, IEEE Comp. Science Press, Silver Spring, MD (1986) 649-652.

[9] K. Hwang and F. Briggs, Parallel processing and computer architecture, *Mc Graw Hill* (1984).

[10] INMOS, The transputer data book, Press Ltd, Bath (1989).

[11] D.E.Knuth, *The art of computer programming*, vol. 2, Addison Wesley Publishing Co., second edition, (1969).

[12] J.D. Lipson, *Elements of algebra and algebraic computing*, Addison Wesley P. Co. (1981).

[13] M.T. McClellan, The exact solution of systems of linear equations with polynomials coefficients, *J. of ACM* 20 (1973) 563-588.

[14] B. Plateau et A.Touzène, Mesures de performances des communications du Méganode à 128 transputers, *La lettre du transputer et des calculateurs distribués* 7 (1990) 73-77.

[15] Y. Robert, B. Tourancheau and G.Villard, Data allocation strategies for the Gauss and Jordan algorithms on a ring of processors, *Information Processing Letters* 31 (1989) 21-29.

[16] F.Roch, *Calcul Formel et parallélisme : forme normale d'Hermite, méthodes de calcul et Parallélisation*, Thèse de l'Instistut National Polytechnique de Grenoble, France (1990).

[17] J.L. Roch, F. Siebert, P. Sénéchaud et G. Villard, Computer Algebra on a MIMD machine, *1988 International Symposium on Symbolic and Algebraic Computation*, Roma, Italy, in *SIGSAM Bulletin*, ACM press, vol. 23 n°1 (1989).

[18] Y. Saad, Gaussian elimination on hypercubes, in M. Cosnard et al. eds. *Parallel Algorithms and Architectures* , North-Holland, Amsterdam, (1986) 5-18.

[19] Telmat Informatique, T.node overview, TN/doc/1_02/3.2 (1990).

[20] H. Takahasi and Y. Ishibashi, A new method for exact calculation by a digital computer, *Information Processing in Japan*, 1 (1961) 28-42.

[21] G. Villard, *Calcul Formel et parallélisme : résolution de systèmes linéaires*, Thèse de l'Institut National Polytechnique de Grenoble, France (1988).

[22] G.Villard, *Chinese remaindering on a MIMD parallel computer*, submitted for publication in *Journal of Symbolic Computation*.

[23] D.M. Young and R.T. Gregory, *A Survey of Numerical Mathematics*, Vol. 2, Dover Publications, Inc. (1988).