Mise en oeuvre d'algorithmes numériques sur un hypercube

Brigitte Vital

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 975

Programme 2

MISE EN OEUVRE D'ALGORITHMES NUMERIQUES SUR UN HYPERCUBE

Brigitte VITAL

Février 1989

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Mise en oeuvre d'algorithmes numériques sur un hypercube

Implementation of some numerical algorithms on a hypercube

Brigitte Vital
IRISA/INRIA, Campus de Beaulieu
35042 Rennes Cedex
FRANCE

Publication Interne n° 450 - Janvier 1989 - 28 Pages

Résumé

Nous présentons la mise en oeuvre sur un hypercube (iPSC-INTEL) de trois algorithmes numériques : l'algorithme de Gram-Schmidt Modifié (MGS) (orthonormalisation d'un ensemble de vecteurs) et deux algorithmes de décomposition d'une matrice en valeurs singulières (SVD). Deux aspects sont abordés :
la répartition des données sur les processeurs :
dans le cas de l'algorithme MGS où une topologie fixe est choisie (en l'occurence: anneau) , nous essayons d'évaluer la taille optimale d'un bloc attribué à un processus .
le choix de la topologie :
dans le cas de la SVD nous avons implémenté deux algorithmes s'adaptant à deux topologies distinctes: anneau de processeurs (méthode d'Hestenes) et grille de processeurs (rotations de Jacobi).
Nous présentons des résultats expérimentaux qui confirment une analyse théorique des performances où l'on suppose que calculs et communications ne se recouvrent pas.

Abstract

We present the implementation of three numerical algorithms on an INTEL-hypercube : modified Gram-Schmidt algorithm (MGS) (orthonormalisation of a group of vectors) and two algorithms for the singular value decomposition of a real matrix (SVD) . Two different aspects are examined :
data allocation :
for the MGS algorithm where a fixed topology is chosen (ring) , we try to find the optimal size of a group of vectors to be allocated to one process .
choice of topology :
for the SVD we have implemented two algorithms fitting on two different topologies : ring of processors (Hestenes method) and grid of processors (Jacobi rotations).
We present some experimental results which agree with the theorical analysis of the performance where we assume no overlapping between communication and computation.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

I Introduction- Présentation de l'hypercube

Nous étudions ici le comportement sur un hypercube de trois algorithmes numériques: orthogonalisation par la méthode de Gram-Schmidt modifiée sur un anneau et deux algorithmes de décomposition en valeurs singulières, l'un pour un anneau, l'autre pour une grille.

Un hypercube est un calculateur multiprocesseur, à mémoire distribuée: chaque processeur a sa propre mémoire locale et ne peut échanger des données avec ses voisins que par envoi et réception de messages. Dans un cube de dimension d, chaque noeud a d voisins (d liens physiques); on peut considérer le cube de manière à obtenir des topologies différentes qui conservent la proximité physique des noeuds voisins ; ceci en renumérotant les processeurs à l'aide des codes de Gray. Nous avons utilisé ici deux topologies particulières: anneau et grille à deux dimensions (voir figure 1).

Dans le cube la rapidité des échanges dépend essentiellement de la distance physique entre les noeuds; il est donc difficile d'évaluer les temps d'exécution; on peut dire que le temps d'envoi d'un message entre deux noeuds voisins est de la forme:

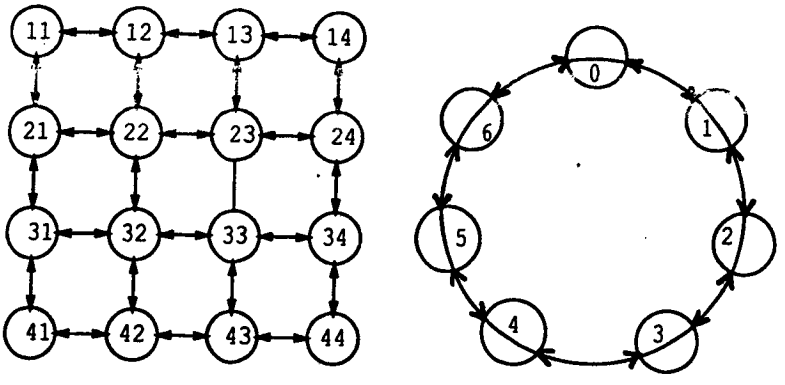
$$t_c = \beta + n\,\tau_c$$

où n est la longueur du message; β est le start-up et τ_c le debit.
Nous faisons de plus l'hypothèse que les calculs et les communications ne se recouvrent pas.
Par la suite on essaie de montrer l'importance dans le comportement des algorithmes du rapport élémentaire :

$$\frac{\tau_c}{\tau_a} = \frac{\textit{temps de communication}}{\textit{temps de calcul}}$$

C'est ce qui a été déjà fait pour les algorithmes de Gauss et Jordan par Y.Robert, B.Tourancheau et G.Villard [5] , qui ont travaillé sur l'hypercube FPS de Grenoble; sur cet hypercube ce rapport a été évalué à : 14.13 .
Ici les expériences ont été réalisées sur l'IPSC (1 et 2) présent à l'IRISA ; nous avons évalué le rapport à environ: 0.3 ; le déséquilibre est moins important que sur le FPS qui possède des processeurs vectoriels, d'où un rapport beaucoup plus faible pour l'IPSC.

figure 1 : topologies grille et anneau



grille carree

anneau

II Algorithme de Gram-Schmidt modifié sur un anneau

1) L'algorithme

Soit à orthonormaliser m vecteurs de longueur n : $(v_1,v_2,...,v_m)$. L'algorithme sequentiel de Gram-Schmidt Modifié (GSM) entre dans une catégorie d'algorithmes (dont Gauss) qui s'écrivent :

```
pour i = 1 , m faire:
    [ i , i ]
    pour j = i + 1 , m faire:
      [ i , j ]
    fait.
fait.
```

Dans le cas de GSM , [i , i] signifie normaliser le vecteur numéro i , c.a.d: $v_i := v_i / \|v_i\|$, et [i , j] signifie corriger le vecteur numéro j par le vecteur numéro i , c.a.d: $v_j := v_j - (v_i , v_j) v_i$.
Au contraire de l'algorithme de Gauss, toutes les tâches [i,j] (avec $i>j$) comportent le même nombre d'opérations.
Le nombre total d'opérations est :

$$N = 2\, m^2\, n + m\, n + m$$

Le temps séquentiel est donc:

$$T_{seq} = (2\, m^2\, n + m\, n + m)\, \tau_a$$

si τ_a est le temps moyen d'une operation flottante sur un processeur.

2) Implémentations possibles sur un anneau de processeurs

Supposons que l'on ait exactement p = m/2 processeurs ou (m+1)/2 si m est impair; ces processeurs sont disposés en anneau et numérotés de 0 à : p - 1 (voir figure 1); la figure 2 montre comment on peut implémenter en parallèle l'algorithme GSM dans ces conditions, en faisant en sorte qu'un processeur ne communique qu'avec ses deux voisins. Les vecteurs circulent dans l'anneau, d'abord dans un sens, puis, à peu près à la moitié du temps, dans l'autre sens; disons qu'il y a un 'aller' et un 'retour'. On peut écrire l'algorithme parallèle (GSMP) ainsi:

```
d := numero du noeud dans l'anneau + 1
( Aller )
pour  j = d  jusqu'à  m - d  faire
     recevoir le vecteur j
     [ d , j ]
     sauf si j = d , envoyer le vecteur j au successeur
( Retour )
pour  j = d + 1  jusqu'à  m - d + 1  faire
     sauf si j = m - d + 1, recevoir le vecteur j
     [ j , m - d + 1 ]
```

envoyer le vecteur j au prédecesseur

Par la suite nous supposons que m est pair pour simplifier.
Si on a plus de 2p vecteurs à traiter on peut opérer de deux manières:

a) entrelacée

On imagine que l'on a m/2 processeurs fictifs pour réaliser l'algorithme parallèle précédent et on répartit le travail sur les processeurs existants;

si $m/2=qp+s$ *avec* : $0 < s \leq p$, le noeud 0 par exemple prend successivement le rôle des processeurs fictifs: 0 , p , 2p , ... , qp à l'aller puis des processeurs fictifs: qp , (q-1)p , ..., 0 au retour. La figure 3 illustre cette répartition dans le cas de 12 vecteurs et 4 processeurs.

b) par blocs

les m vecteurs sont répartis en 2p groupes équilibrés; l'algorithme parallèle est le même que dans le cas m = 2p mais cette fois:

[i , i] signifie appliquer GSM séquentiel au groupe i

[i , j] signifie corriger successivement tous les vecteurs du groupe j par rapport à tous ceux du groupe i .

noeuds		0	1	2	3
	1 -->	[1,1]			
aller:	2 -->	[1,2]			
envoi	3 -->	[1,3]	[2,2]		
des	4 -->	[1,4]	[2,3]		
vecteurs	5 -->	[1,5]	[2,4]	[3,3]	
	6 -->	[1,6]	[2,5]	[3,4]	
	7 -->	[1,7]	[2,6]	[3,5]	[4,4]
	8 -->	[1,8]	[2,7]	[3,6]	[4,5]
	1 <--				
retour:	2 <--	[2,8]	[3,7]	[4,6]	[5,5]
reception	3 <--	[3,8]	[4,7]	[5,6]	
des	4 <--	[4,8]	[5,7]	[6,6]	
vecteurs	5 <--	[5,8]	[6,7]		
orthonormalisés	6 <--	[6,8]	[7,7]		
	7 <--	[7,8]			
	8 <--	[8,8]			

figure 2
GSM sur un anneau
de m/2 processeurs
(m = 8)

noeuds	0	1	2	3
1 -->	[1,1]			
2 -->	[1,2]			
-	[1,3]	[2,2]		
-	[1,4]	[2,3]		
-	[1,5]	[2,4]	[3,3]	
-	[1,6]	[2,5]	[3,4]	
-	[1,7]	[2,6]	[3,5]	[4,4]
-	-	-	-	-
-	-	-	-	-
-	-	-	-	-
12 -->	[1,12]	[2,11]	[3,10]	[4,9]
1 <--	[5,5]			
	[5,6]			
	[5,7]	[6,6]		
	[5,8]	[6,7]		
	[6,8]	[7,7]		
	[7,8]			
	[8,8]			
2 <--	[2,12]	[3,11]	[4,10]	[5,9]
3 <--	[3,12]	[4,11]	[5,10]	[6,9]
-	-	-	-	-
-	-	-	-	-
-	-	-	-	[9,9]
-	-	-	-	
-	-	-	[10,10]	
-	-	-		
-	-	[11,11]		
-	-			
12 <--	[12,12]			

figure 3
méthode entrelacée pour
12 vecteurs sur 4 processeurs

Nous avons essayé les deux méthodes sur l'hypercube de l'IRISA; en pratique c'est la méthode entrelacée qui a toujours été la plus rapide sur l'IPSC1 et l'IPSC2 comme le montrent par exemple les résultats suivants (figure 4):

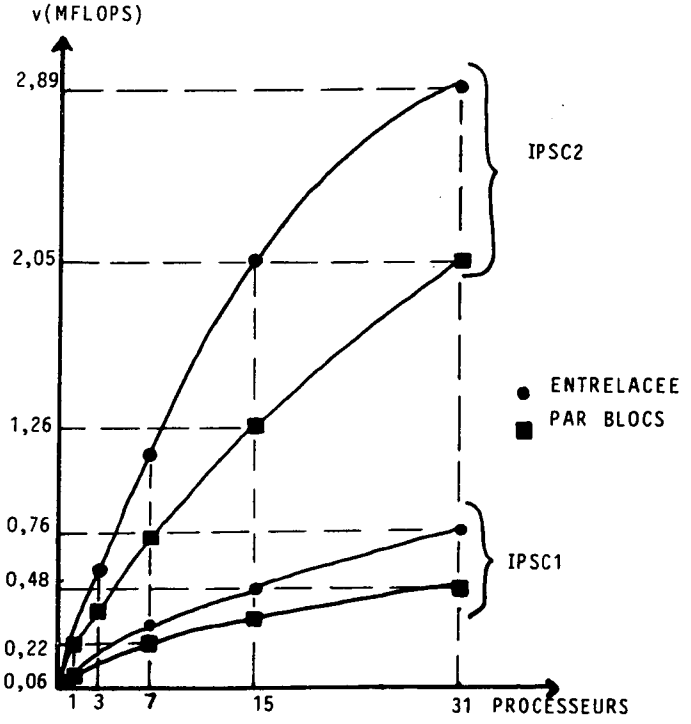


figure 4
vitesse en fonction du nombre de processeurs (m = n = 200)

En fait dans la méthode entrelacée l'efficacité est meilleure que dans celle par blocs, par contre il y a plus de communication ;

c'est la valeur du rapport $\frac{\tau_c}{\tau_a}$, et dans une moindre mesure celle de $\frac{\beta}{\tau_a}$, qui permet de choisir l'une ou l'autre sur une machine comme nous allons essayer de le montrer.

Pour étudier l'algorithme, nous envisageons par la suite une méthode intermédiaire dont les deux précédentes sont des cas particuliers ; elle consiste à former des groupes de vecteurs (comme dans la méthode par blocs) mais en nombre supérieur au double du nombre de processeurs (comme dans la méthode entrelacée).

3) évaluation du temps et efficacité. Meilleure répartition des vecteurs

Imaginons que les m vecteurs sont divisés en blocs égaux mais qu'il y a plus de 2p blocs; cela revient à utiliser la méthode entrelacée en remplaçant les vecteurs par les blocs de vecteurs (voir figure 3) ; pour m et p fixés, la relation entre le nombre de vecteurs dans chaque bloc: r et le nombre de processeurs fictifs à considérer: nproc est:

$$m = 2\,r\,nproc$$

pour r=1 on retrouve la méthode entrelacée
pour r=m/2p on retrouve la méthode par blocs
Ce sont les deux cas extrêmes; le problème est de trouver, pour m et p fixés, la valeur de r dans l'intervalle: [1 , m/2p] qui rendra l'efficacité maximum.

Soient q et l les uniques entiers tels que:

$$nproc=pq+s \text{ , } avec : 0 < s \leq p$$

on peut interpréter ainsi: les noeuds 0 , 1, ... , $s - 1$ font le travail de q + 1 processeurs fictifs, les autres noeuds de q seulement et sont donc inactifs pendant un certain temps(fin de 'l'aller' et début du 'retour').

Le processeur 0 est le premier à commencer et le dernier à finir; c'est donc lui que l'on observe. On peut d'autre part remarquer qu'il n'y a théoriquement pas de temps mort.

Evaluons le temps total : T_p, somme du temps de calcul : T_a et du temps de communications : T_c.

le noeud 0 normalise: $2(q+1)$ blocs de vecteurs et corrige: $\sum_{i=0}^{q}(2\times 2nproc-3-4ip)$ blocs ; on remarque qu'une normalisation d'un bloc s'accompagne soit d'une réception de ce bloc (à l'aller), soit d'un envoi de ce bloc (au retour), tandis qu'une correction est toujours précédée d'une réception et suivie d'un envoi. Nous sommes donc amenés à introduire les quantités suivantes:

t_{nor} : temps pour une normalisation de bloc + un transfert
t_{cor} : temps pour une correction de bloc + un envoi + une réception

Avec ces notations:

$$T_p = 2\,(\,q + 1\,)\,t_{nor} + \left[\sum_{i=0}^{q}\,(\,2\times 2\,nproc - 3 - 4\,i\,p\,)\right] t_{cor}$$

On suppose pour simplifier que nproc est multiple de p, c.a.d que : s=0 (de toutes façons c'est dans ce cas que l'efficacité est la meilleure car il n'y a pas de processeurs inactifs); on a donc:
q+1 = nproc/p = m/2pr et on obtient :

$$T_p = \frac{m}{pr}\,t_{nor} + \left[\frac{m^2}{2pr^2} + (2p-3)\,\frac{m}{2pr}\right] t_{cor}$$

Il reste à évaluer t_{nor} et t_{cor} :

pour orthonormaliser r vecteurs il y a : $2rn + rn + r$ opérations et pour corriger un groupe de r vecteurs par un autre il y en a : $4r^2n$; comme nous supposons la longeur n des vecteurs "infinie", nous négligeons maintenant la latence : β ; en effet les messages sont de longueur : t = nr et on a : $\beta/t \ll \tau_c$; (par exemple, sur l'IPSC2 : β=3,2.10^{-4} s et τ_c = 1,4.10^{-6} s / mot de 32 bits). Donc:

$$t_{nor} = \left(2\,r^2n + rn + r\right)\tau_a + rn\tau_c$$

$$t_{cor} = 4r^2n\,\tau_a + 2rn\tau_c$$

n étant supposé grand, on néglige le terme indépendant de n dans t_{nor} et t_{cor}, d'où ;

$$t_{nor} = n\left[(2\,r^2 + r)\tau_a + r\tau_c\right]$$

$$t_{cor} = n\left[4r^2\,\tau_a + 2r\tau_c\right]$$

L'efficacité est définie par le rapport: T_{seq}/pT_p ; nous avons choisi de comparer le temps sur p processeurs à celui que demanderait l'algorithme sur une machine séquentielle ayant la même puissance qu'un noeud de l'hypercube ; nous ne comptons donc aucune communication dans le temps séquentiel :

$$T_{seq} = (2m^2n + mn)\,\tau_a$$

D'où l'efficacité en fonction de m,n,p et r:

$$e_p = \frac{A}{B + C\,r + \dfrac{D}{r}} = f\,(r)\ ,\ avec:$$

$$A = 2m + 1\ ,\ B = 2m + 1 + \frac{\tau_c}{\tau_a}\,(2p - 2)\ ,\ C = (4p - 4)\ ,\ D = \frac{\tau_c}{\tau_a}\,m$$

Une étude de la fonction f montre que l'efficacité est maximum pour:

$$r_0 = \sqrt{\frac{D}{C}} = \sqrt{\frac{\tau_c\,m}{(\,4\,p - 4\,)\,\tau_a}}$$

remarque:
Cette formule est valable quand n est "infini" ; en pratique nous sommes limités par la taille de la mémoire et les valeurs de m et n essayées sont telles que les termes contenant la latence ne peuvent pas vraiment se négliger, en particulier parceque nous avons transféré chaque bloc vecteur par vecteur : dans notre implémentation le temps de transfert d'un bloc est en fait: $r\,(\beta + n\tau_c)$; en tenant compte de ces termes on obtient :

$$r_0 = \sqrt{\frac{\tau_c\,mn + \beta\,m}{(\,4\,p - 4\,)\,n\,\tau_a}}$$

exemple:
Sur l'IPSC2 nous avons mesure le rapport: τ_c / τ_a : il est environ de 0.3 ; nous avons aussi estime la latence pour les longs messages : $\beta / \tau_a = 69.1$;
Pour p=7 et: m=n=336, la théorie prévoit donc une efficacité maximale pour: $r_o = 2.66$; la figure 5 représente les résultats que nous avons obtenus sur cet hypercube; on voit que le temps est minimum pour r = 3 ; c'est bien ce qu'on espérait. Presque tous les essais réalisés ont donné une valeur de 2 ou 3 pour r_o.

D'autre part, on peut retrouver les efficacités pour les méthodes par blocs et entrelacée en prenant les valeurs extrêmes de r : r=m/2p et r=1 ; voici quelques exemples d'efficacités prévues par la théorie pour 200 vecteurs de longueur 200 sur l'IPSC2 :

p	1	3	7	15	31	63
efficacite(par blocs)	1.00	0.59	0.53	0.50	0.48	0.46
efficacite(entrelacee)	1.00	0.74	0.72	0.67	0.60	0.49

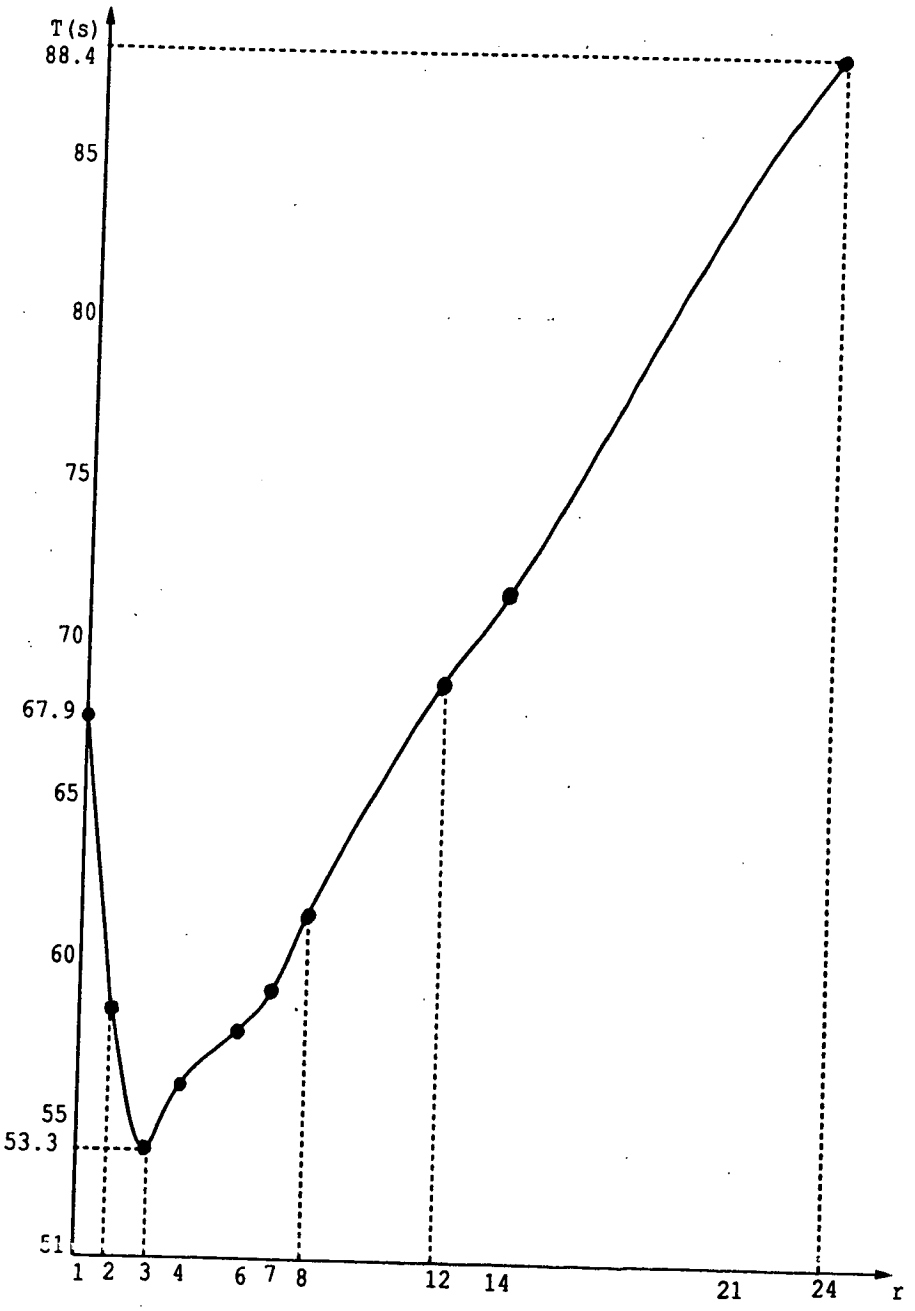
Les résultats obtenus sur l'IPSC2 ont été un peu moins bons; de toutes façons, on voit que dans la méthode entrelacée l'efficacité est meilleure que dans celle par blocs; En faisant tendre m et n vers l'infini on obtient les valeurs asymptotiques de l'efficacité :
pour la méthode par blocs :

$$(e_p)_\infty = \frac{1}{2 - \frac{1}{p}}$$

pour la méthode entrelacée :

$$(e_p)_\infty = \frac{1}{1 + 0.5 \frac{\tau_c}{\tau_a}}$$

figure 5
temps en fonction de la taille des blocs sur 7 processeurs pour m=n=336 sur l'IPSC2



III Etude du comportement sur l'hypercube de deux algorithmes de décomposition d'une matrice en valeurs singulières (ou SVD)

1) SVD d'une matrice réelle

Etant donnee une matrice réelle n x m : A, avec: $n \geq m$,
sa décomposition en valeurs singulières peut se définir ainsi: il existe trois matrices $U : n \times m$, $V : m \times m$, *et* $S : m \times m$ telles que :

$$A = U\,S\,V^t \text{ , avec :}$$

$$U^t\,U = V^t\,V = I_m \text{ et: } S = diag(s_1, \cdots, s_m) \text{ , } s_i \geq 0$$

Les éléments diagonaux de S sont appelés valeurs singulières de A; leurs carrés sont les valeurs propres de A^tA et les plus grandes valeurs propres de AA^t; les colonnes de V (resp. de U) sont les vecteurs propres associés de A^tA (resp. de AA^t).
La SVD est très utilisée en calcul numérique (problème aux moindres carrés, résolution de systèmes linéaires rectangulaires, optimisation...); la méthode classique, celle qui est utilisée dans EISPACK [6], consiste à bidiagonaliser la matrice par des transformations de Householder puis à calculer les valeurs singulières de la matrice bidiagonale par l'algorithme QR ; Une variante [7], bien adaptée aux matrices ayant beaucoup plus de lignes que de colonnes, consiste à réduire d'abord la matrice à une forme triangulaire supérieure avant de bidiagonaliser. Cette méthode est très mal adaptée aux calculateurs parallèles, ce qui explique que l'on étudie actuellement des algorithmes nouveaux spécialement conçus pour ces machines.
Nous avons teste deux de ces algorithmes qui utilisent deux topologies différentes, en les implémentant sur l'hypercube de l'IRISA , et nous les avons comparés : l'un est proposé par Brent et Luk [1], l'autre par Brent, Luk et Van Loan [2].

2) SVD par la méthode d'Hestenes sur un anneau de processeurs

2-1) La méthode d'Hestenes

La méthode d'Hestenes consiste à générer une matrice orthogonale mxm: V , telle que la matrice H = AV ait ses colonnes deux à deux orthogonales; il ne reste ensuite qu'à normer à l'unité les colonnes de H pour obtenir:

$$H = A\,V = U\,S \text{ , } S = diag(s_1, \cdots, s_m) \text{ , } U^tU = I_m$$

où s_i est la norme du i^{eme} vecteur colonne de H.
La matrice V est obtenue comme un produit de rotations planes; une rotation: $R(p,q,\theta)$ est définie par un couple d'indices (p,q), avec p<q qui donne le plan de la rotation et un angle: θ; R ne diffère de l'identité que par les éléments suivants :

$$R_{p,q} = -\,R_{q,p} = \sin\theta \text{ ; } R_{p,p} = R_{q,q} = \cos\theta$$

Multiplier à droite une matrice par R revient à remplacer les colonnes p et q par une combinaison linéaire de ces colonnes, les autres colonnes n'étant pas touchées; on peut donc simultanément traiter deux rotations de couples d'indices disjoints. Par la suite l'angle sera choisi pour rendre les colonnes p et q de la matrice H orthogonales de la manière suivante:

Algorithme 1 :

soit u et v les vecteurs à orthogonaliser,

$$\alpha = u^t u \ , \ \beta = v^t v \ , \ \gamma = u^t v$$

$$\xi = \frac{\beta - \alpha}{2\gamma} \ , \ t = \frac{sign(\xi)}{|\xi| + \sqrt{1 + \xi^2}}$$

$$\cos\theta = \frac{1}{\sqrt{1 + t^2}} \ , \ \sin\theta = t \cos\theta$$

Si les deux vecteurs sont déjà orthogonaux on prendra comme rotation l'identité; on dira que deux vecteurs u et v sont orthogonaux si:

$$\frac{| u^t v |}{\|u\|_2 \, \|v\|_2} \leq \varepsilon$$

où la précision ε est à choisir par l'utilisateur. Un balayage complet de la matrice consiste à traiter tous les couples (p,q) avec: p < q; l'ordre habituel pour l'algorithme séquentiel est l'ordre lexicographique :

(1,2),(1,3), ... ,(1,m),(2,3),...,(2,m),...,(m-1,m)

On effectue un certain nombre de balayages de la matrice jusqu'à ce que les colonnes soient jugées orthogonales deux à deux à la précision choisie ; l'algorithme séquentiel est:

Algorithme (HS) :

$H = A$; $V = Id_m$
tant que l'orthogonalité est insuffisante faire:
pour p = 1 , m faire:
pour q = p + 1 , m faire:
-calculer θ par l'algorithme 1 pour que les colonnes d'indices p et q de $HR(p,q,\theta)$ soient orthogonales
$-H = H\,R\,(p\,,q\,,\theta)$
$-V = V\,R\,(p\,,q\,,\theta)$

(calcul des valeurs singulières et de U)
pour i = 1 , m faire:

$$s_i = \| h_i \|_2 \ ; \ u_i = \frac{1}{s_i} . h_i$$

fait.

La complexité de cet algorithme peut se calculer de la manière suivante : le calcul d'un angle de rotation demande 6 n + 12 opérations. Si on veut tous les vecteurs propres on doit mettre à jour après chaque rotation 2 colonnes de longueur n (celles de H) et 2 de longueur m (celles de V), ce qui demande 6 m + 6 n opérations; il y a m(m-1)/2 rotations pour un balayage;
D'autre part on doit normaliser les m colonnes de H à la fin du calcul, soit m(3n+1) opérations; donc si S est le nombre de balayages, le nombre d'opérations pour l'algorithme séquentiel est:

$$N = S \left[\frac{m\,(m-1)}{2} \,(12\,n + 6\,m + 12) \right] + m\,(3\,n + 1)$$

Pour un seul balayage le temps séquentiel est donc:

$$T_{seq} = (3\,m^3 + 6\,m^2\,n - 3\,m\,n + 3\,m^2 - 5\,m)\,\tau_a$$

2-2) Implémentation sur un anneau

Brent et Luk ont proposé une implémentation de cette méthode sur un réseau linéaire systolique de p=m/2 processeurs [1]; voici comment. On dispose d'un anneau de processeurs numérotés de 0 à p-1 et on a m = 2 p vecteurs; il s'agit de faire un balayage complet de la matrice en un certain nombre d'étapes de telle sorte qu'à chaque étape les processeurs se voient attribuer des couples d'indices disjoints qu'ils peuvent traiter simultanément; la figure 6 montre la répartition adoptée par Brent et Luk.

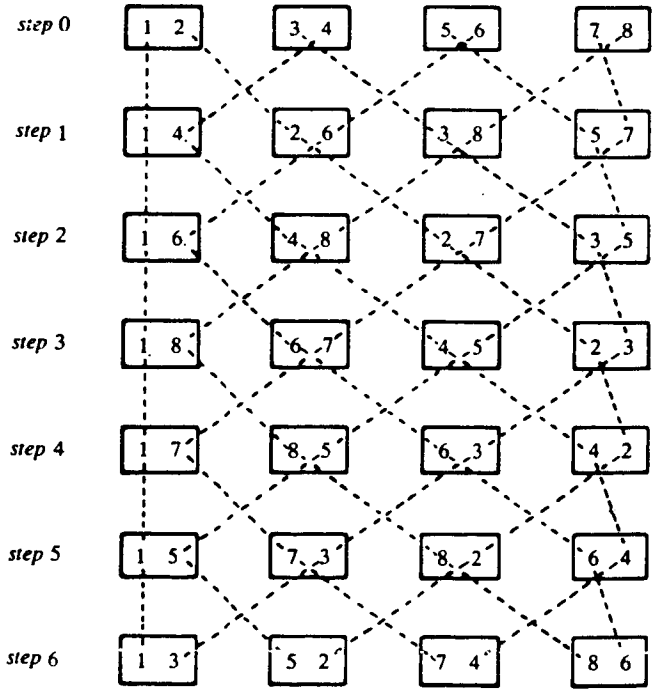


figure 6

méthode d'Hestenes sur un anneau de m/2 processeurs (m = 8)

Ici un balayage se fait en 2p -1 étapes et à la fin de chacune d'elles les processeurs

échangent leurs vecteurs avec leurs deux voisins; en fait cela revient à appliquer aux m colonnes de la matrice une permutation P telle que:

$$P^{2p-1} = Id$$

la permutation choisie est:

(1,2,3,4,5,6,...,2k-1, 2k ,....,m-2,m-1, m)
(1,3,5,2,7,4,...,2k+1,2k-2,...,m-4, m ,m-2)

Si maintenant, ce qui est le cas sur un hypercube où le nombre de processeurs est limité ($p \leq 64$ sur l'IPSC) , on a plus de 2p vecteurs colonnes on peut constituer 2p blocs de vecteurs consécutifs ; soit r le nombre de vecteurs dans un bloc ; l'algorithme par blocs est le même que dans le cas m = 2p (figure 6), si on définit ainsi le traitement du couple d'indices (p,q):

*à l'étape 1 de chaque balayage :

orthogonaliser deux à deux, à l'aide de l'algorithme 1, les 2r vecteurs appartenant à la réunion des groupes p et q , c.a.d leur appliquer un balayage de HS.

* aux autre étapes:

orthogonaliser successivement, à l'aide de l'algorithme 1, tous les couples de vecteurs de la forme : (u,v) où u est dans le groupe p et v est dans le groupe q.

Ainsi en 2p - 1 étapes on effectue bien un balayage complet de la matrice A.

2-3) Estimation du temps et de l'efficacité pour un balayage complet

On voit que les processeurs sont pratiquement synchrones: ils commencent et terminent en même temps; le temps d'un processeur quelconque donnera donc le temps global.
Soit r le nombre de vecteurs dans un bloc; on a donc: $m = 2\,p\,r$. (si m est impair on commence par rajouter à la matrice une colonne de 0); au cours d'un balayage il y a:

- 2p-1 étapes de calcul
- 2p-2 échanges

Comme les processeurs effectuent tous le même nombre d'opérations, le temps de calcul est: $T_a = T_{seq} / p$; soit:

$$T_a = \frac{(3\,m^3 + 6\,m^2\,n - 3\,m\,n + 3\,m^2 - 5\,m)\,\tau_a}{p}$$

A chaque échange le processeur envoie 2r vecteurs de longueur n, 2r de longueur m, et il en reçoit autant; en négligeant la latence, le temps de communication est donc:

$$T_c = 2\,(2\,p - 2) \times 2\,r\,(m + n)\,\tau_c$$

En tenant compte du fait que: r = m/2p, le temps total est: $T_p = T_c + T_a$ et l'efficacité:

$$e_p = \frac{3m^3 + 6m^2n - 3mn + 3m^2 + O(m)}{3m^3 + 6m^2n - 3mn + 3m^2 + (2p - 2)(2\,m^2 + 2\,mn)\,\frac{\tau_c}{\tau_a} + O(m)}$$

On voit que:

$$\lim_{n\to\infty} e_p = \frac{1}{1 + \frac{4(p-1)}{3(2m-1)} \cdot \frac{\tau_c}{\tau_a}}$$

2-4) Resultats expérimentaux

Avec Catherine Ratsivalaka [3], nous avons implémenté et testé cet algorithme sur l'IPSC pour des matrices carrées ou rectangulaires ; nous mesurons ici [figures 7,8 et 9] les temps d'exécution sur l'IPSC2 pour un balayage complet avec calcul des vecteurs propres, suivi du calcul des valeurs singulières ; (dans notre programme le calcul des vecteurs propres est optionnel mais de toutes façons on doit aussi calculer U pour obtenir uniquement les valeurs singulières). Notons que pour n=m=176 il faut 11 balayages pour arriver à la convergence avec la précision : $\varepsilon=10^{-6}$ et pour n=300 , m=64 il en faut 8 .
La figure 7 montre la linéarité du temps d'exécution par rapport à la longueur n des vecteurs colonnes pour un nombre m de vecteurs fixé.

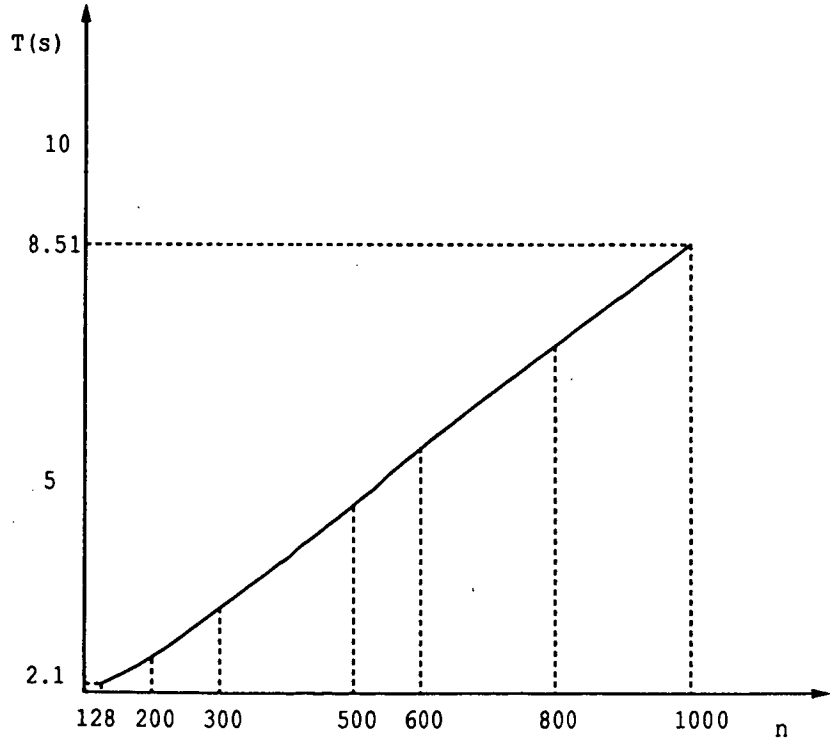


figure 7
Temps en fonction de n sur un balayage pour m=128 (IPSC2 , 64 processeurs)

Vitesse sur un balayage en fonction du nombre
de processeurs pour deux matrices différentes

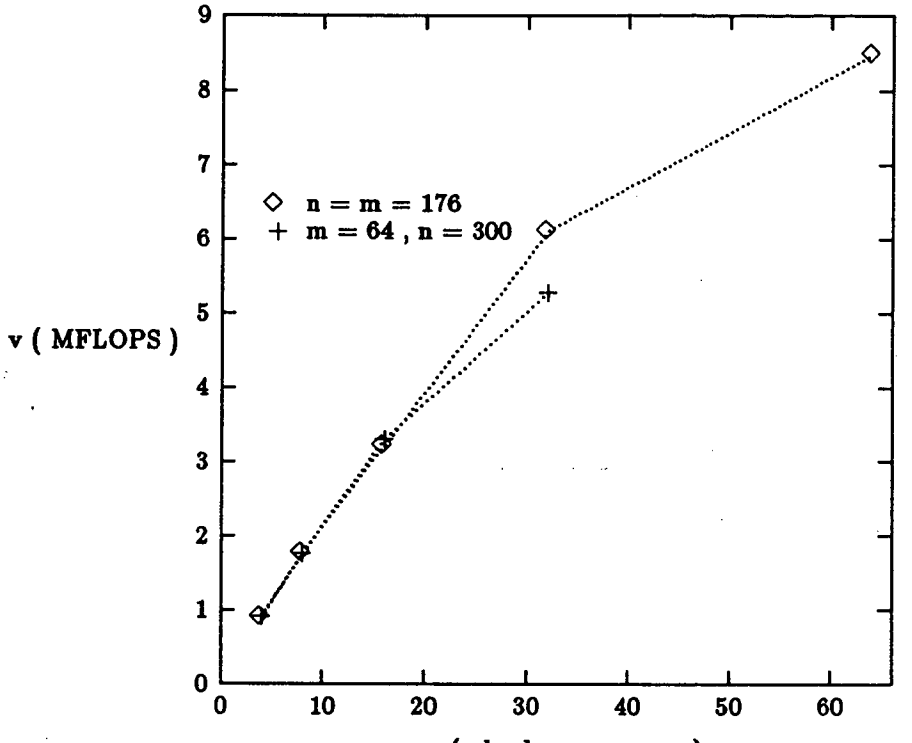


figure 8

efficacité sur un balayage en fonction
du nombre de processeurs pour deux matrices différentes

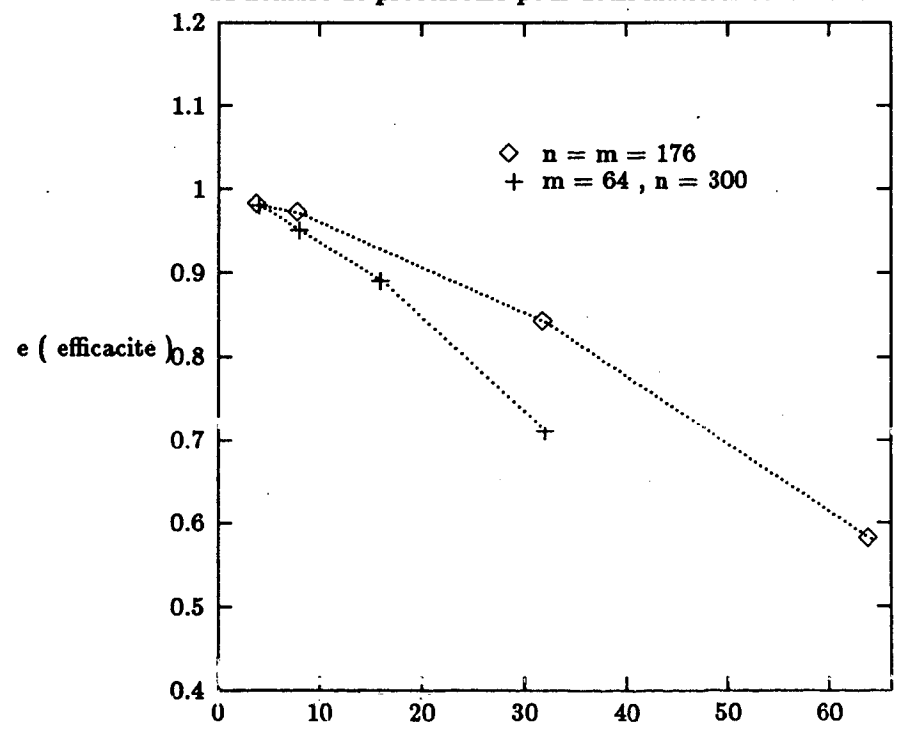


figure 9

3) SVD sur une grille de processeurs

La méthode suivante a été proposée par Brent, Luk et Van Loan [2]; il s'agit de décomposer une matrice réelle en valeurs singulières en utilisant une grille carrée de processeurs ; cette topologie impose à la matrice d'être carrée , sans quoi il faut commencer par la rendre carrée en lui ajoutant des zéros ; cette méthode est donc à éviter pour des matrices ayant deux dimensions très différentes ; cependant elle offre la possibilité de ne rechercher que les valeurs singulières sans calculer U et V, alors que dans l'algorithme précédent l'obtention des valeurs singulières passe nécessairement par le calcul de U .
On essaie ici de paralléliser un algorithme séquentiel, proposé par Forsythe et Henrici [4], et expliqué ci-dessous (paragraphe 3-1) ; C. Bischof et C. Van Loan [8] ont aussi implémenté une version parallèle de cette méthode sur un anneau de processeurs : il s'agit en fait du même algorithme que celui qui est présenté ici pour une grille, si on attribue à chaque processeur de l'anneau le rôle d'une colonne de processeurs de la grille.

3-1) L'algorithme séquentiel

Le principe est de multiplier la matrice A, à droite et à gauche par des rotations planes distinctes pour annuler à la fois les deux éléments: $a_{p,q}$ et $a_{q,p}$; les rotations ont des angles distincts mais sont dans le même plan (p,q).
Les deux angles θ_1 et θ_2 qui doivent annuler les elements d'indices (p,q) et (q,p) de $R(p,q,\theta_1)^t\, A\, R(p,q,\theta_2)$ se calculent ainsi:

Algorithme USVD:

$$w=a_{p,p}\ ,\ z=a_{q,q}\ ,\ x=a_{p,q}\ ,\ y=a_{q,p}$$

(Symétrisation):

- calculer θ ,p,q et r tels que:

$$\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}w & x\\ y & z\end{bmatrix}=\begin{bmatrix}p & q\\ q & r\end{bmatrix}$$

(Diagonalisation du résultat et fin)

- trouver θ_2 , d_1 *et* d_2 tels que:

$$r^t\begin{bmatrix}p & q\\ q & r\end{bmatrix}r=\begin{bmatrix}d_1 & o\\ 0 & d_2\end{bmatrix}\ ,\ avec:\ r=\begin{bmatrix}\cos\theta_2 & \sin\theta_2\\ -\sin\theta_2 & \cos\theta_2\end{bmatrix}$$

- $\theta_1 = \theta + \theta_2$

On réalise un balayage complet de A en traitant successivement tous les couples (p,q), $p < q$ suivant l'ordre habituel; à la fin d'un balayage on regarde si la nouvelle matrice est assez diagonale au sens suivant suivant:

$$off(A) < \varepsilon\|A_0\|_F\ ,\ avec:\ off(A) = \sum_{p<q\ ou\ q<p} a_{p,q}^2$$

où A_0 est la matrice initiale; si nécessaire on fait un nouveau balayage. L'algorithme séquentiel est:

$U = V = Id_n$
jusqu'à ce que la matrice soit assez diagonale faire:
pour p = 1 , n faire:
pour q = p+1 , n faire:
- calculer θ_1 et θ_2 pour que les éléments
d'indices (p,q) et (q,p) de: $R(p,q,\theta_1)^t\ A\ R(p,q,\theta_2)$ soient nuls
- $A := R(p,q,\theta_1)^t\ A\ R(p,q,\theta_2)$
- $U = U\ R(p,q,\theta_1)$
- $V = V\ R(p,q,\theta_2)$

La mise à jour de la matrice A consiste à remplacer les lignes et colonnes p et q par des combinaisons linéaires d'elles mêmes et les autre éléments sont inchangés;

La complexité de l'algorithme peut se calculer ainsi :
Au cours d'un balayage il y a n(n-1)/2 transformations; à chaque transformation il y a:
- un passage dans USVD: 52 opérations
- 3 mises à jours de 2 colonnes (A,U,V): 18n
- 1 mise à jour de 2 lignes(A): 6n

soit: (n(n-1)/2)x(24n+52) opérations. D'où le temps pour un balayage séquentiel sans test de convergence :

$$T_{seq} = (12\ n^3 + 14\ n^2 - 26\ n)\ \tau_a$$

3-2) Implémentation sur une grille de processeurs

- Répartition des données:

On dispose d'une grille carrée pxp de processeurs (figure 1); on suppose que la taille de A est un multiple de 2p: n=2pr ; la matrice est divisée en sous-matrices carrées de taille 2r comme le montre le dessin ci-dessous; ces sous-matrices sont réparties sur les processeurs: le processeur $P_{i,j}$ (i^{eme} ligne, j^{eme} colonne de la grille), reçoit la matrice : $A_{i,j}$.

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

division de la matrice en p^2 sous-matrices (p=4)

La sous-matrice : A_{loc} de taille 2r affectée à un processeur est elle-même divisée en 4 blocs carrés de taille r :

$$A_{loc} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$$

Si on veut calculer les vecteurs propres U et V, chaque processeur doit en plus contenir deux

autres matrices de taille 2r : U_{loc} et V_{loc} (les matrices U et V sont réparties de la même façon que A sur la grille de processeurs);

- L'algorithme parallèle:

Les noeuds diagonaux, $P_{i,i}$, agissent différemment des autres: ce sont eux qui calculent les angles des rotations à appliquer à la matrice, les autres noeuds ne font que mettre à jour la matrice A (éventuellement U et V).
Au cours d'une étape, chaque noeud diagonal applique à sa matrice locale : A_{loc} l'algorithme séquentiel, mais, après chaque rotation, il diffuse horizontalement et verticalement les paramètres des rotations (cosinus et sinus); plus exactement, la rotation de gauche se propage horizontalement et celle de droite verticalement (voir figure 10). Ainsi les processeurs de la même rangée mettent à jour les lignes adéquates de leur matrice locale, et ceux de la même colonne les colonnes adéquates.
si on calcule les vecteurs propres il faut propager aussi verticalement la rotation de gauche, car elle sert à mettre à jour U;
Après une étape on regarde si la matrice est assez diagonale; en fait comme le test est coûteux en communication, on a choisi de le faire uniquement après un nombre fixé d'étapes: I, que l'on choisit (dans la pratique I est choisi égal à 2p-1 ou à un multiple de 2p-1 ; ce choix s'explique par la suite).
Si la matrice n'est pas assez diagonale, on effectue une permutation sur les lignes et les colonnes de la matrice de taille 2p x 2p formées par les blocs; cette permutation P, la même que dans la méthode d'Hestenes, est telle que chaque noeud intérieur échange ses données avec ses 4 voisins diagonaux (voir figures 10 et 11); on réalise alors une autre étape mais cette fois un noeud diagonal traite seulement les couples (p,q) tels que :

$$1 \le p \le r \, , \, r+1 \le q \le 2r$$

Comme: $P^{2p-1} = Id$, on réalise ainsi un balayage complet de la matrice A en: 2p-1 étapes.

L'algorithme parallèle (avec calcul de U et V) s'écrit:

tant que A n'est pas "assez" diagonale faire:

pour i=1,I faire:

Si le noeud est diagonal alors:
pour (p,q) dans Δ faire:
- calculer θ_1 et θ_2 pour que les elements d'indices p et q de $r(p,q,\theta_1)^t A_{loc} r(p,q,\theta_2)$ soient nuls
- diffuser horizontalement: $\cos\theta_1$, $\sin\theta_1$
- diffuser verticalement: $\cos\theta_1$, $\sin\theta_1$, $\cos\theta_2$, $\sin\theta_2$
- $A_{loc} = r(p,q,\theta_1)^t A_{loc} r(p,q,\theta_2)$
- $U_{loc} = U_{loc} r(p,q,\theta_1)$
- $V_{loc} = V_{loc} r(p,q,\theta_2)$

sinon:

pour (p,q) dans Δ faire:
- recevoir: $\cos\theta_1$, $\sin\theta_1$ et les diffuser horizontalement
- recevoir: $\cos\theta_1^{bis}$, $\sin\theta_1^{bis}$, $\cos\theta_2$, $\sin\theta_2$ et les diffuser verticalement
- $A_{loc} = r(p,q,\theta_1)^t A_{loc}\, r(p,q,\theta_2)$
- $U_{loc} = U_{loc}\, r(p,q,\theta_1^{bis})$
- $V_{loc} = V_{loc}\, r(p,q,\theta_2)$

envoyer a,u et v et recevoir les nouvelles valeurs suivant la procédure d'échange

avec: $\Delta = \left\{ (p\,,\,q)\,,\, p+1 \le q \le 2r \right\}$ pour la première itération de la boucle {i}, et: $\Delta = \left\{ (p\,,\,q)\,,\, 1 \le p \le r\,,\, r+1 \le q \le 2r \right\}$ pour les autres.

- Nombre d'opérations:

Soit n_a le nombre d'opérations pour une transformation ($n_a = 24n+52$) ; soit: T le nombre de tests et: I le nombre d'étapes entre deux tests; le nombre total d'opérations est:

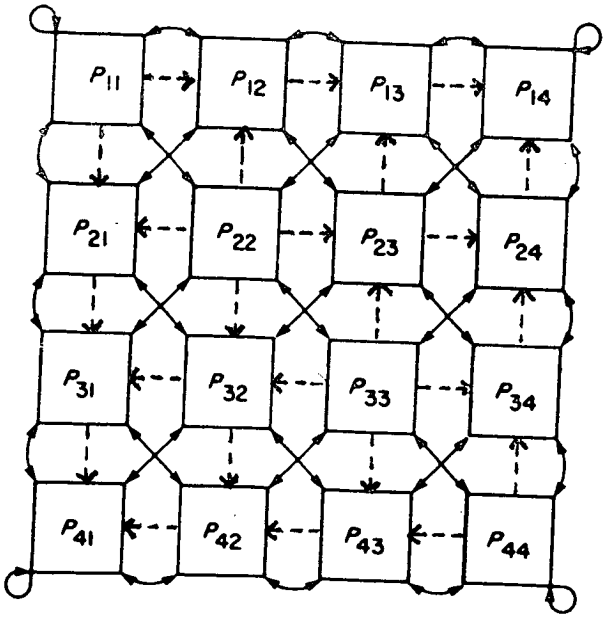
$$N = T\,p \left[\frac{2r(2r-1)}{2} + (I-1)\,r^2 \right] n_a + T\,2n^2$$

comme n=2pr, pour un balayage complet avec un test (I=2p-1 , T=1), on obtient:

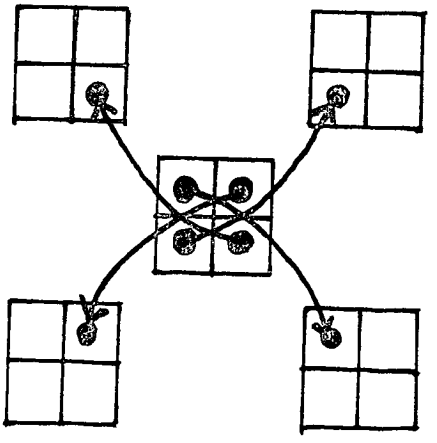
$$N = (12n^3 + 14n^2 - 26n) + 2n^2$$

figures 10 et 11

diffusion des paramètres des rotations
échanges des blocs



destination des 4 blocs pour un noeud intérieur



3-3) Estimation du temps et de l'efficacité pour un balayage complet suivi d'un test

Supposons que chaque noeud propage à ses voisins les cosinus et sinus des rotations ainsi :
horizontalement : d'abord vers la droite puis vers la gauche, et *verticalement :* d'abord vers le haut puis vers le bas ; alors le processeur : $P_{p,1}$, dans le coin en bas à gauche, est le dernier à commencer ses calculs de mise à jour ; calculons donc le temps nécessaire à ce processeur pour terminer son étape.
Pour tenir compte des temps morts dûs à ce que les autres noeuds ont plus de communications à faire on peut supposer que ce processeur propage lui aussi les paramètres des rotations à des voisins fictifs ; d'autre part bien que seulement deux réels se propagent horizontalement, on peut supposer qu'il y en a autant que verticalement, soit : 4 , car tout noeud attend de toutes façons un message horizontal et un message vertical avant de calculer, d'où des temps morts.
Nous prenons donc le modèle simplifié d'une grille infinie dans laquelle se propageraient à la même vitesse des messages de même longueur horizontalement et verticalement.

Nous avons deux types de messages dans cet algorithme :

- *des messages courts :* dans la diffusion des cosinus et sinus des rotations, on envoie et on reçoit 4 réels ; nous définissons donc : τ_s : temps de transfert de 4 mots de 32 bits entre 2 noeuds voisins.
- *des messages longs :* dans les échanges de blocs entre voisins diagonaux ; comme les noeuds sont à distance : 2 , le temps pour un tel message de longueur t est : $2\,t\,\tau_c$, si on néglige la latence : β .

Faisons le bilan pour le noeud choisi ; au cours d'un balayage il y a : 2p-1 étapes et : 2p-2 échanges ; le noeud traite : n(n-1)/2p rotations ; donc :

pour les communications et temps morts:

* somme des temps morts avant réception des premiers paramètres (à chaque étape : diffusion de 4 réels sur un côté de la grille) :

$$(2p-1)\left[\,2(p-2)\,\tau_s\,\right] = O(n)$$

* somme des temps morts dûs au surcroît de travail des processeurs diagonaux (à chaque rotation : 1 passage dans USVD) :

$$\frac{n(n-1)}{2p}\,52\tau_a$$

* échanges diagonaux (à chaque échange : 1 envoi + 1 réception) :

$$(2p-2)\,\frac{4n^2}{p^2}\,\tau_c$$

* diffusion des paramètres des rotations (à chaque rotation : 2 réceptions + 2 envois) :

$$\frac{n(n-1)}{2p}\,4\tau_s$$

pour les calculs :

* mise à jour de A,U,V (à chaque rotation : 6 colonnes + 2 lignes) :

$$\frac{n(n-1)}{2p} \times \frac{24n}{p}\ \tau_a$$

* calcul de somme partielle des carrés pour le test final :

$$\frac{2\,n^2}{p^2}\ \tau_a$$

D'où l'efficacité :

$$e_{p^2} = \frac{T_{seq}}{p^2\,(T_a + T_c)} = \frac{12n^3+16n^2+O(n)}{12n^3-10n^2+26pn^2+2pn^2\ \frac{\tau_s}{\tau_a}+(8p-8)\ n^2\frac{\tau_c}{\tau_a}+O(n)}$$

A la limite : $\lim_{n\to\infty} e_{p^2} = 1$.

Sur l'IPSC2 on a estimé le rapport τ_s / τ_a à environ : 40 . par exemple : pour n=176 et p=8 le modèle prévoit :

$$e_{64} = 0{,}72$$

et les tests ont donné :

$$e_{64} = 0{,}69$$

Pour finir, voici quelques résultats d'expériences sur l'IPSC2 (figures 12 et 13) : les temps sont mesurés sur un balayage complet suivi d'un test de convergence ; notons qu'il faut 7 balayages pour diagonaliser une matrice d'ordre n=176 avec une précision de : $\varepsilon=10^{-6}$.

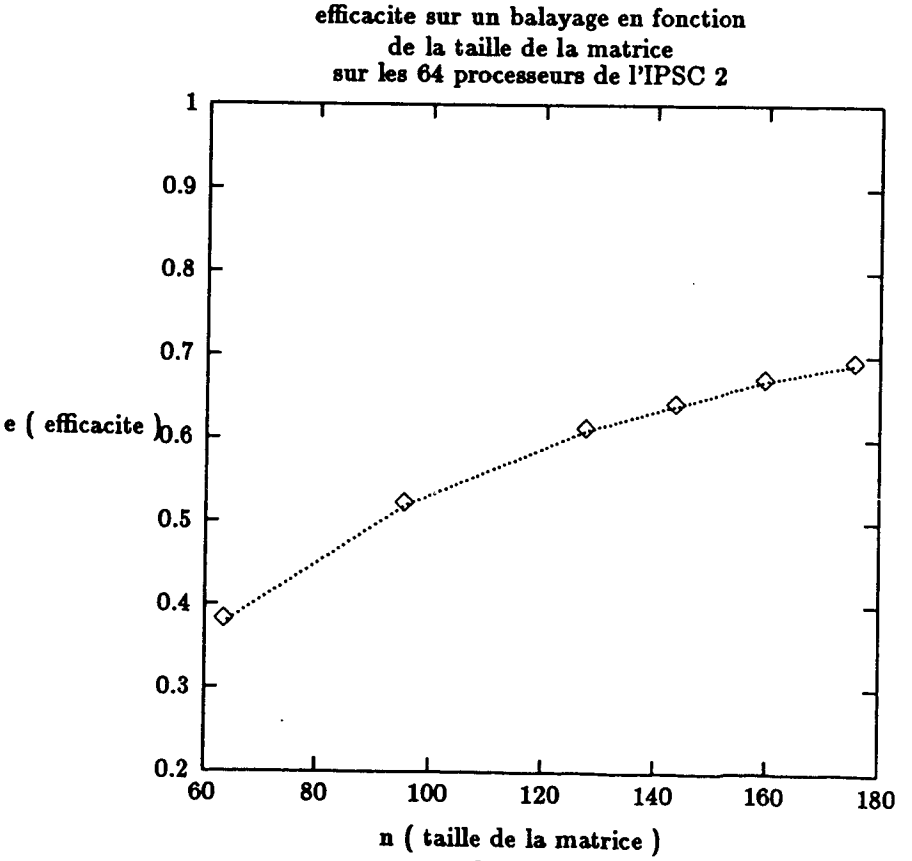


figure 12

processeurs	4	16	64
temps sur un balayage (s)	64.63	18.55	5.84
acceleration	3.7	13.0	41.3
efficacite	0.93	0.81	0.64
vitesse (MFLOPS)	0.56	1.95	6.19

figure 13
Performances pour une matrice carrée d'ordre n=144
en fonction du nombre de processeurs (IPSC2)

4) Comparaison des deux algorithmes de SVD

Pour pouvoir comparer les 2 algorithmes, nous avons fait les tests sur des matrices carrées. En effet l'algorithme grille est très mal adapté aux matrices rectangulaires que l'on doit rendre carrées en leur rajoutant des zéros; ce qui donne une très mauvaise efficacité ! Cependant il offre l'avantage de pouvoir calculer les valeurs singulières sans les vecteurs propres, alors que dans l'algorithme anneau, U est automatiquement calculé.
Pour une matrice 176x176 et sur 64 processeurs, nous avons obtenu sur l'IPSC2 une vitesse de: 6,65 MPLOPS sur la grille et une vitesse de: 8.47 MFLOPS sur l'anneau. Le nombre d'opérations est plus élevé pour l'algorithme grille ($12\ n^3$) que pour l'anneau ($9\ n^3$) ; par contre si on ne veut pas les vecteurs propres, il faut $6\ n^3$ opérations pour les deux algorithmes.

Quand il s'agit de calculer les vecteurs propres, toutes les expériences que nous avons faites montre qu'il faut choisir l'anneau comme topologie ; les figures 14 et 15 donnent les temps pour les deux algorithmes dans ce cas : les temps sont mesurés sur un seul balayage suivi de 1 test (figure 15) ou sur le nombre nécessaire de balayages pour arriver à la convergence avec la précision:

$$\varepsilon = 10^{-6}$$

et avec I=2p-1 (figure 14).

Par contre si les vecteurs propres ne sont pas demandés, les deux algorithmes réalisent le même nombre d'opérations mais le temps de communication est toujours plus grand pour la grille que pour l'anneau (à peu près le double) , ceci en particulier à cause des communications diagonales dans la grille de processeurs; aussi les performances restent dans ce cas meilleures pour la topologie anneau, bien que les différences soient un peu moins marquées (figure 16).

Nous avons d'autre part comparé l'algorithme anneau (qui est sûrement dans ce cas le plus rapide des deux algorithmes étudiés car on fournit les vecteurs propres) avec la routine SVD de Eispack; ii ressort des expériences faites qu'il faut au moins 8 processeurs sur l'IPSC1 pour égaler les performances de Eispack pour des matrices de taille comprises entre 50 et 200 (voir figure 17).

Temps total en fonction du nombre
de processeurs pour les deux algorithmes (IPSC2)

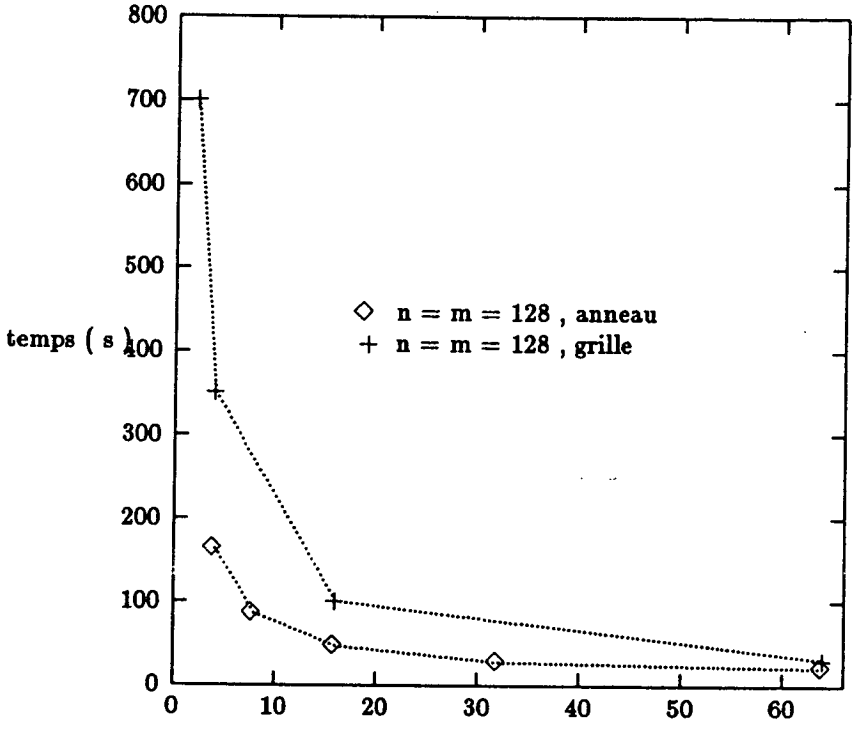


figure 14

temps sur un balayage en fonction de la taille
des matrices carrées, sur 16 et 64 processeurs (IPSC2)

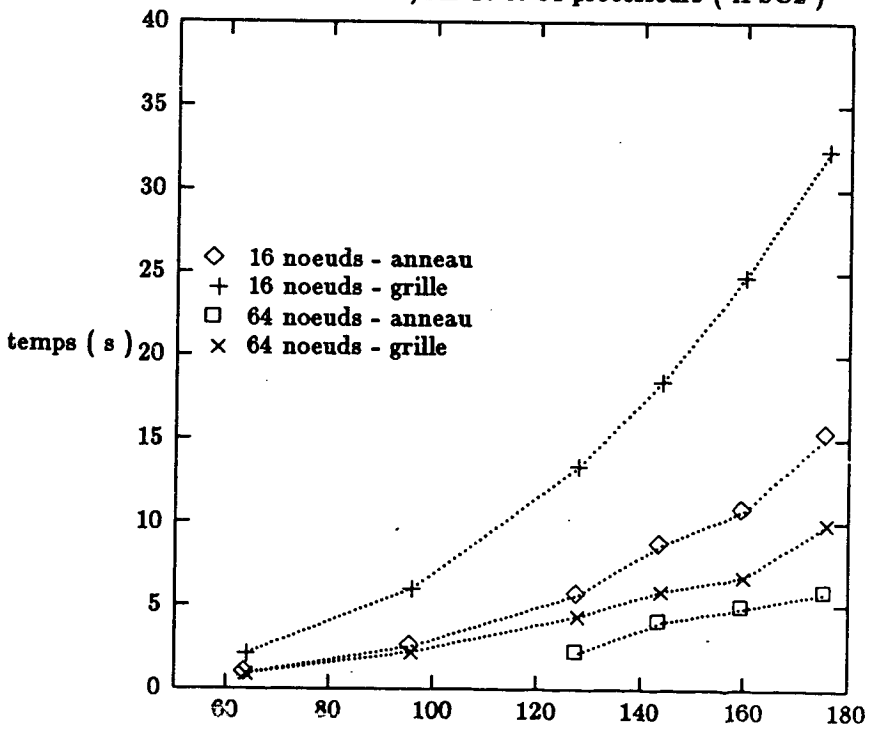


figure 15

nb. de processeurs	4	16	64
temps (s):			
grille	61.32	18.47	6.33
anneau	35.53	9.79	3.60
efficacite:			
grille	0.92	0.76	0.55
anneau	0.99	0.89	0.61
vitesse :			
grille	0.54	1.81	5.27
anneau	0.92	3.35	9.10

figure 16
Performances des 2 algorithmes pour une matrice carrée d'ordre n=176
sur un balayage sans calculer les vecteurs propres (IPSC 2)

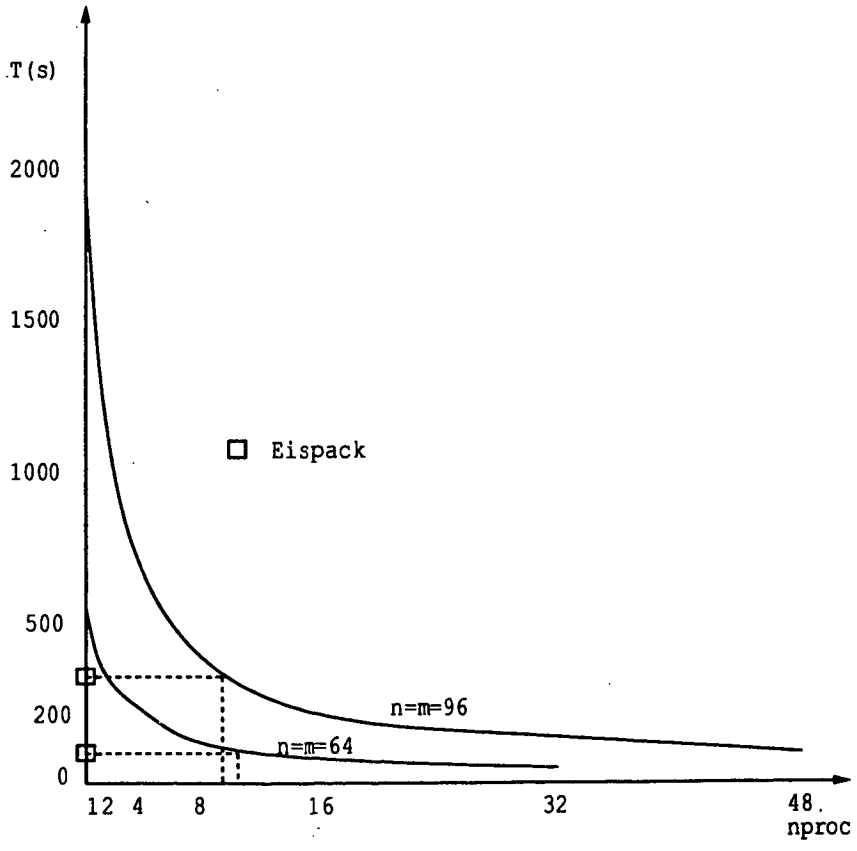


figure 17
Comparaison sur l'IPSC1 de l'algorithme anneau avec Eispack

BIBLIOGRAPHIE

[1] P.Brent et Franklin T.luk, The solution of Singular-Value and symmetric eigenvalue problems on multiprocessor arrays, SIAM J. Sci Stat. Comput., vol. 6, no. 1, (1985)

[2] P.Brent, Franklin T.Luk et C.Van Loan, Computation of the Singular Value Decomposition using mesh-connected processors, journal of VLSI and computers systems, volume 1, number 3

[3] Catherine Ratsivalaka, Implémentation sur un anneau de processeurs d'un algorithme de décomposition en valeurs singulières, rapport stage de magistère (1988)

[4] G.E.Forsythe et P.Henrici, The cyclic Jacobi method for computing the principal values of a complex matrix, Trans.Amer.Math.Soc.94, pp. 1-23 (1960)

[5] Y.Robert, B.Tourancheau, G.Villard, Data allocation strategies for the Gauss and Jordan algorithms on a ring of processors, Rapport technique IMAG Grenoble no. 40 , (septembre 1988)

[6] B.S.Garbow,J.M.Boyle,J.J.Dongarra et C.B.Moler, Matrix eigensystem routines-EISPACK Guide Extension, Springer-Verlag, Berlin (1977)

[7] T F.Chan, An improved algorithm for computing the singular value decomposition , ACM Transactions on Mathematical Software, vol.8, No.1, pp.72-83 (mars 1982)

[8] C.Bischof et C.Van Loan, Computing the singular value decomposition on a ring of array processors, Large Scale Eigenvalue Problems, Elsevier Science Publishers B.V. (North-Holland), (1988)

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 438 - A PROPOS DE LA RESOLUTION D'UN SYSTEME LINEAIRE DANS UN CORPS FINI : ALGORITHMES ET MACHINES PARALLELES
Hervé LE VERGE, Patrice QUINTON, Yves ROBERT, Gilles VILLARD
22 Pages, Novembre 1988.

PI 439 - ALPHA DU CENTAUR : A PROTOTYPE ENVIRONMENT FOR THE DESIGN OF PARALLEL REGULAR ALGORITHMS
Pierrick GACHET, Patrice QUINTON, Christophe MAURAS, Yannick SAOUTER
20 Pages, Novembre 1988.

PI 440 - CONSTRUCTION METHODIQUE D'UN ALGORITHME REPARTI DE DETECTION DE LA TERMINAISON
Jean-Michel HELARY, Michel RAYNAL
18 Pages, Décembre 1988.

PI 441 - LES GRAPHES A MOTIF
Didier CAUCAL
46 Pages, Décembre 1988.

PI 442 - CAUSAL TREES
Philippe DARONDEAU, Pierpaolo DEGANO
44 Pages, Décembre 1988.

PI 443 - TROIS IMPLANTATIONS DU RECUPERATEUR DE MEMOIRE DE LA MACHINE MALI
Michel LE HENAFF, Hervé SANSON
118 Pages, Décembre 1988.

PI 444 - ANALYSE FACTORIELLE LISSEE ET ANALYSE FACTORIELLE DES DIFFERENCES LOCALES
Brigitte ESCOFIER, Habib BENALI
34 Pages, Décembre 1988.

PI 445 - MULTISCALE STATISTICAL SIGNAL PROCESSING
Michèle BASSEVILLE, Albert BENVENISTE
16 Pages, Décembre 1988.

PI 446 - MODELES STATISTIQUES TEMPS-ECHELLE EN TRAITEMENT DU SIGNAL
Michèle BASSEVILLE, Albert BENVENISTE
60 Pages, Décembre 1988.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique