



Ecole CNRS :
INFORMATIQUE
SCIENTIFIQUE POUR LE
CALCUL

Systemes Linéaires Denses

Pierre RAMET
(projet ScAIApplix INRIA Bordeaux)

Plan

- Algorithmes de factorisation
 - LU, Cholesky
 - Parallélisme
- ScaLAPACK





Algorithmes de factorisation

Séquentiels et Parallèles

Introduction

- Résoudre $Ax=b$
- Ce matin, A dense !!!
- \rightarrow non adapté aux pb E.F ou D.F.
- Distributions régulières
- Parallélisme « simple »
- Elimination de Gauss
- Pb du pivotage (complet, partiel, statique)

Contexte général

Directe
 $A = LU$

Itérative
 $y' = Ay$

Non
symétrique

**Pivoting
LU**

**GMRES,
BiCGSTAB,
...**

Symétrique
définie
positive

Cholesky

**Conjugate
gradient**



Convergence / Rapidité / Stockage

Problématique du parallélisme

- Distribution des données
 - Pb cache, pipeline d'instructions
- Expression du parallélisme
 - Algorithmique
 - Informatique
- Complexité séquentielle souvent \neq complexité parallèle
- Architectures
 - Mémoire partagée (BLAS SMP) : threads
 - Mémoire distribuée : MPI (PVM)
 - Mixte (cluster de nœuds SMP ou NUMA)
 - GPU

Principe de la factorisation

- $Ax=b$ (complexité $O(n^3)$)
- b peut être un multi-second membre
- Plusieurs solutions à trouver pour un même système A (boucle en temps)
- Factoriser A (complexité $O(n^3)$)
- Résolution en $O(n^2)$
- Exemple LU : produit de 2 matrices triangulaires (diagonale $U=1$)
 - $LUx=b$
 - $Ly=b$ puis $Ux=y$

Résolution de systèmes triangulaires ($Lx=b$) (x écrase b)

- Algorithme (bloc) ligne (**distribution colonne**)

résoudre $L_{11}X_1 = B_1$

pour i de 2 à p faire

$$B_i = B_i - [L_{i1} \quad L_{i,i-1}] \times [X_1 \quad X_{i-1}]^T$$

résoudre $L_{ii}X_i = B_i$

- Algorithme (bloc) colonne (**distribution ligne**)

pour j de 1 à p-1 faire

résoudre $L_{jj}X_j = B_j$

$$[B_{j+1} \quad B_p]^T = [B_{j+1} \quad B_p]^T - [L_{j+1,j} \quad L_{p,j}]^T \times X_j$$

résoudre $L_{pp}X_p = B_p$

Exemple matrice 4x4 blocs avec 2 processeurs

■ Processeur 1

■ Processeur 2

■ Résoudre $L_{11}X_1=B_1$

■ $B_2=B_2-L_{21}X_1$

■ $B_3=B_3-L_{31}X_1$

■ $B_4=B_4-L_{41}X_1$

■ Résoudre $L_{33}X_3=B_3$

■ $B_4=B_4-L_{43}X_3$

■ Résoudre $L_{44}X_4=B_4$

■ Résoudre $L_{22}X_2=B_2$

■ $B_3=B_3-L_{32}X_2$

■ $B_4=B_4-L_{42}X_2$



Elimination de Gauss sans pivotage (A=LU) A inversible

```
... for each column i,  
... zero it out below the diagonal by  
... adding multiples of row i to later rows  
for i = 1 to n-1  
    ... each row j below row i  
    for j = i+1 to n  
        ... add a multiple of row i to row j  
        for k = i to n  
             $A(j,k) = A(j,k) -$   
                 $(A(j,i)/A(i,i)) * A(i,k)$ 
```

En supprimant les calculs redondant on arrive à l'algorithme suivant :

```
for i = 1 to n-1  
    for j = i+1 to n  
         $A(j,i) = A(j,i)/A(i,i)$  ... store L on top of A  
    for k = i+1 to n ... we have also reversed the  
        for j = i+1 to n ... order of the j and k loops  
             $A(j,k) = A(j,k) - A(j,i) * A(i,k)$ 
```

A la fin, A a été écrasé par L et U

Factorisation de Cholesky ($A=LL^t$) A SPD

On modifie l'algorithme précédent tel que $U=L^t$

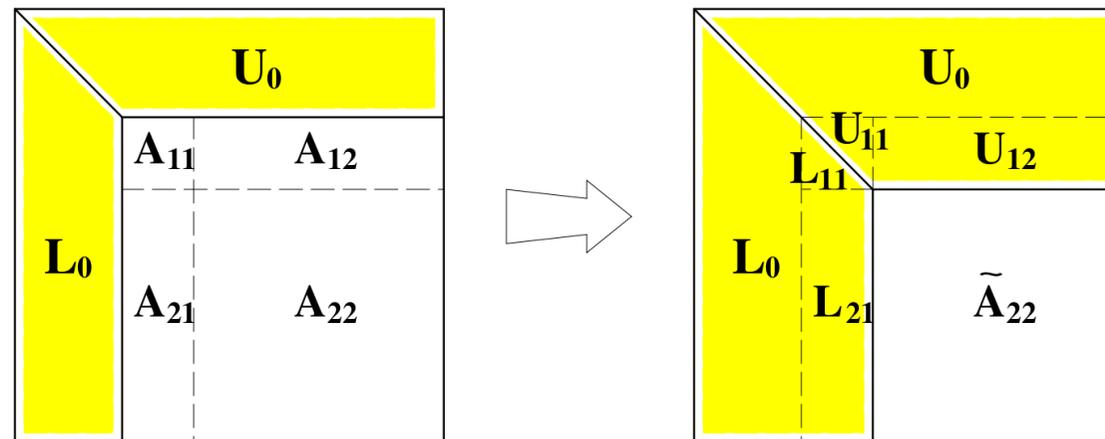
```
for i = 1 to n-1
  A(i,i) = sqrt(A(i,i)) ... if error, A not positive definite
  for j = i+1 to n
    A(j,i) = A(j,i)/A(i,i) ... store L on top of A
    A(i,j) = A(i,j)/A(i,i) ... compute U in place too
  for k = i+1 to n
    for j = i+1 to n
      A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

Comme $U=L^T$ on élimine la partie triangulaire supérieure de A

```
for i = 1 to n-1
  A(i,i) = sqrt(A(i,i)) ... if error, A not positive definite
  for j = i+1 to n
    A(j,i) = A(j,i)/A(i,i) ... store col i of L on top of A
  for k = i+1 to n
    for j = k to n ... update only on and below diagonal
      A(j,k) = A(j,k) - A(j,i) * A(k,i) ... A(k,i)=A(i,k)
```

A=LU

$$\begin{aligned} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} &= P \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} \\ &= P \begin{pmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{pmatrix} \end{aligned}$$



A=LU

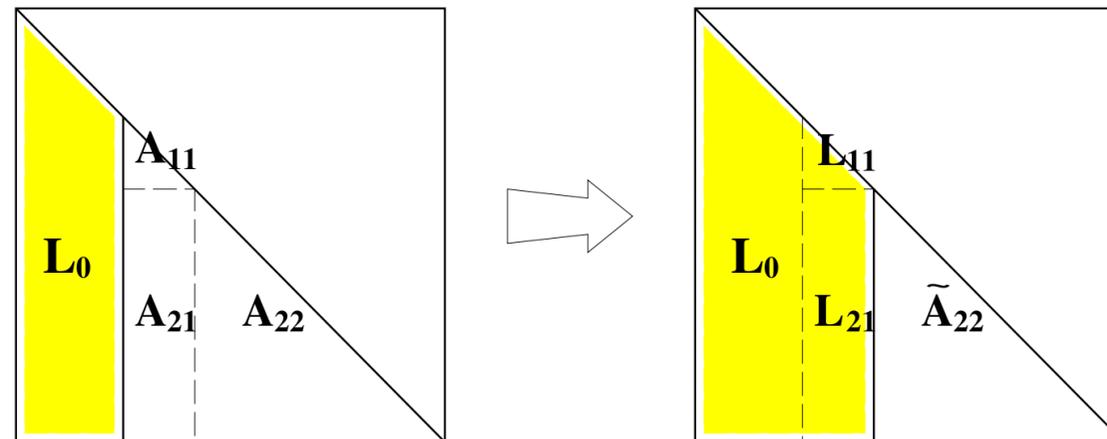
- PDGETF2 : factorisation LU ($m \times n_b$) sur A_{11} et A_{21} [sur P_r]
 - Répéter n_b fois ($i=1..n_b$) :
 - PIDAMAX : recherche du pivot (**absolu**) dans la colonne i
 - PDSWAP : échange la $i^{\text{ème}}$ ligne et celle du pivot
 - PDSCAL : division de la colonne i par le pivot
 - PDGER : **diffusion sur P_r** et mise à jour du reste de la sous-matrice
- PDLASWP : appliquer le pivotage [sur $P_c \times P_r$]
- PDTRSM : **diffusion de L_{11} sur P_c**
$$U_{12} \leftarrow (L_{11})^{-1}A_{12}$$
- PDGEMM : **diffusion de L_{21} et U_{12} sur P_c et P_r**
$$\underline{A}_{22} \leftarrow A_{22} - L_{21}U_{12}$$
$$\dots = L_{22}U_{22}$$

Parallélisme

- Opérations collectives
- Processeurs regroupés dans des échanges par ligne ou colonne
- $P = P_c \times P_r$
- → intérêt d'une bibliothèque dédiée pour des échanges sur une grille de processeurs (BLACS)
- Indépendance / réseau
- Indépendance / bibliothèque de communication (PVM, MPI ...)

$$A = LL^t$$

$$\begin{pmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{pmatrix} \\ = \begin{pmatrix} L_{11}L_{11}^T & L_{11}L_{21}^T \\ L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T \end{pmatrix}$$



$$A = LL^t$$

- **PDPOTF2** : factorisation de Cholesky ($n_b \times n_b$) sur A_{11} [en local]
 - Détection non-définie-positive + diffusion
- **PDTRSM** : diffusion de L_{11} sur P_r
$$L_{21} \leftarrow A_{21}(L_{11}^t)^{-1}$$
- **PDSYRK** : diffusion de L_{21} sur P_c + transposition
$$\underline{A}_{22} \leftarrow A_{22} - L_{21}L_{21}^t$$
$$\dots = L_{22}L_{22}^t$$

Chemin critique

- Partie intrinsèquement séquentielle d'un algorithme parallèle
- Pour les factorisations :
 - Complexité du chemin critique : $O(n^2)$
 - Volume de communication : $O(n^2)$
- Granularité assez fine pour avoir autant de calcul que de communication



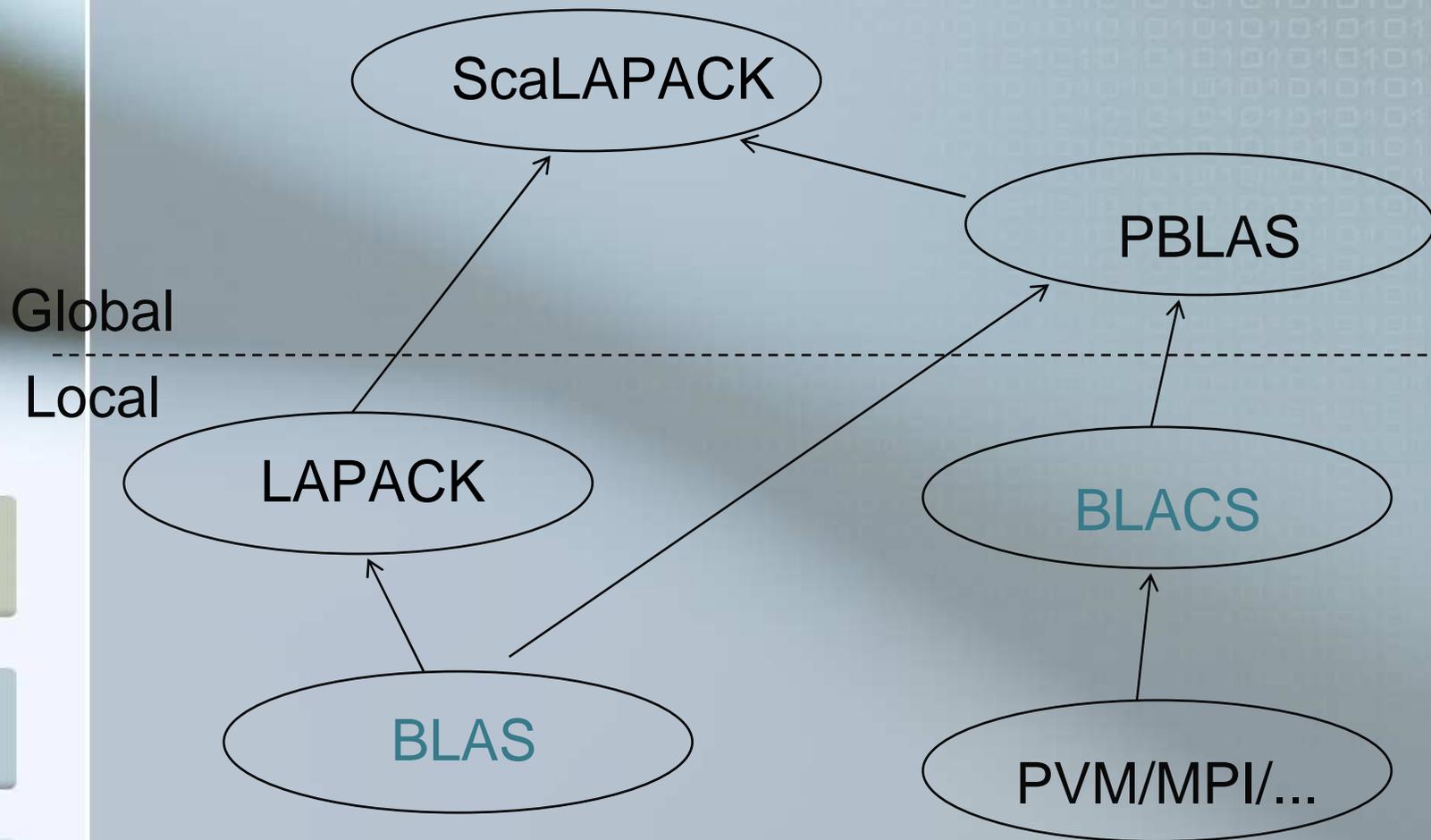
ScaLAPACK

(à partir du tutoriel de netlib.org)

ScaLAPACK

- Jack Dongarra
- = LAPACK parallèle
- Contient :
 - Résolution de systèmes linéaires
 - Recherche valeurs/vecteurs propres
 - Routines de tri ...
- Dérivés constructeurs
 - mkl (intel)
 - acml (AMD)
 - essl (IBM)
 - ...

Structure de ScaLAPACK



But - Porter LAPACK pour des environnements distribués.

- **Efficacité**
 - Moteurs de calcul et de communication optimisés
 - Algorithmes par blocs (Level 3 BLAS) utilise les hiérarchies mémoires
- **Réutilisabilité**
 - Lorsque c'est possible, ré-utilise les algorithmes LAPACK
- **Scalabilité**
 - En fonction de la taille du problème et du nombre processeurs
 - Remplace les algorithmes LAPACK qui ne "scale" pas
- **Portabilité**
 - Dépendances aux architectures reportées sur les BLAS et les BLACS
- **Modularité**
 - Ensembles d'outils pour l'algèbre linéaire : BLAS, BLACS, PBLAS
- **Facile à utiliser**
 - Interface similaire à LAPACK

Description basée Fortran

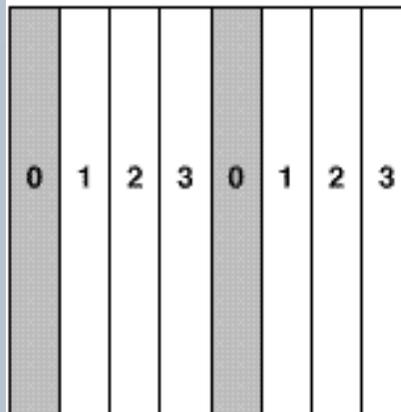
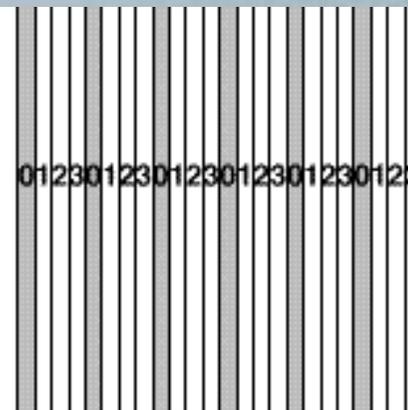
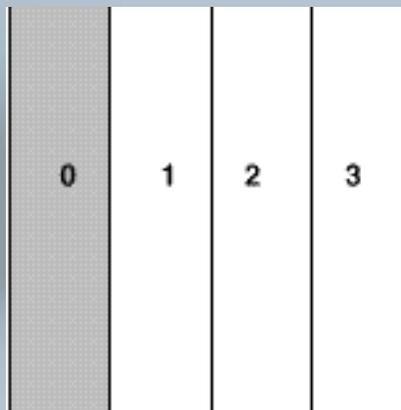
- **A chaque objet global est assigné un "array descriptor".**
 - Contient les informations nécessaires pour assurer la correspondance entre l'objet global et sa distribution sur les processeurs ainsi que l'allocation locale en mémoire
 - Différencié par DTYPE_ (première entrée) du descripteur.
 - Solution souple pour permettre l'ajout de nouveaux schémas de distribution ou de matrices

Array Descriptors

- Support pour plusieurs types de matrices :
 - Matrices denses
 - Matrices bandes and tri-diagonales
 - Matrices "out-of-core"
- L'utilisateur doit distribuer les vecteurs globaux avant d'invoquer une routine ScaLAPACK

Choix d'une distribution

- Block, Cyclic, Block-Cyclic



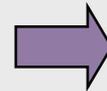
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3



Distribution 2D Bloc-Cyclique

matrice 5x5 partitionnée en blocs 2x2

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}



grille 2x2 de processeurs

A_{11}	A_{12}	A_{15}	A_{13}	A_{14}
A_{21}	A_{22}	A_{25}	A_{23}	A_{24}
A_{51}	A_{52}	A_{55}	A_{53}	A_{54}
A_{31}	A_{32}	A_{35}	A_{33}	A_{34}
A_{41}	A_{42}	A_{45}	A_{43}	A_{44}

Distribution 2D Bloc-Cyclique

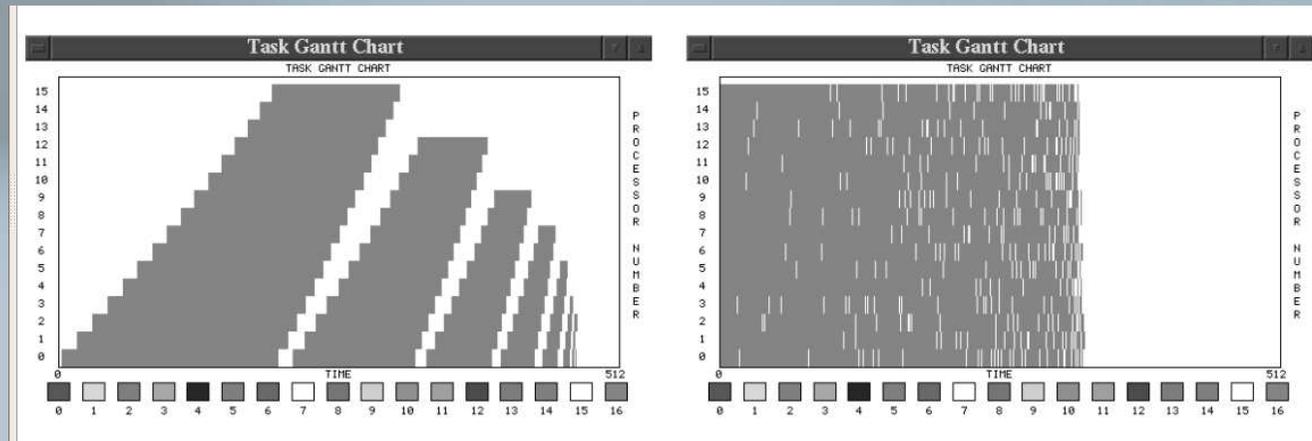
- Soit P le nombre de processeurs avec $P = P_r \times P_c$
- Soit une taille de bloc : $r \times c$
- Soit une matrice de taille : $M \times N$
- L'élément global d'indice (m, n) sera stocké à la position (i, j) dans le bloc (b, d) sur le processeur (p, q) tel que :

$$\{(p, q), (b, d), (i, j)\} = \{(|m/r| \% P_r, |n/c| \% P_c), (||m/r|/P_r|, ||n/c|/P_c|), (m \% r, n \% c)\}$$

Distribution 2D Bloc-Cyclique

- **Assure un bon équilibrage de charge**
--> Performance et Scalabilité
 - Englobe un grand nombre (mais pas tous) de schémas de distribution de données
 - Distribution bloc-pure : $r=|M/P_r|$ et $c=|N/P_c|$
 - Distribution cyclique-pure : $r=c=1$
 - Besoin de routines de redistribution pour passer d'une distribution à une autre

1D versus 2D



Autre solution : 1D avec pipeline ...

Pour une matrice dense

DESC_()	Symbolic Name	Scope	Def init ion
1	DTYPE_A	(global)	Descriptor type DTYPE_A=1 for dense matrices.
2	CTXT_A	(global)	BLACS context handle.
3	M_A	(global)	No. of rows in global array A
4	N_A	(global)	No. of cols. In global array A
5	MB_A	(global)	Blocking factor used to distribute the rows of array A.
6	NB_A	(global)	Blocking factor used to distribute the columns of array A.
7	RSRC_A	(global)	Process row over which the first row of the array A is distributed.
8	CSRC_A	(global)	Process column over which the first column of the array A is distributed.
9	LLD_A	(local)	Leading dimension of the local array.

SEQUENTIAL LU FACTORIZATION CODE

```

DO 20 J = 1, MIN( M, N ), NB
  JB = MIN( MIN( M, N )-J+1, NB )

  Factor diagonal and subdiagonal blocks and test for exact
  singularity.

  CALL DGETF2( M-J+1, JB, A( J, J ), LDA, IPIV( J ),
  $           IINFO )

  Adjust INFO and the pivot indices.

  IF( INFO.EQ.0 .AND. IINFO.GT.0 ) INFO = IINFO + J - 1
  DO 10 I = J, MIN( M, J+JB-1 )
    IPIV( I ) = J - 1 + IPIV( I )
10  CONTINUE

  Apply interchanges to columns 1:J-1.

  CALL DLASWP( J-1, A, LDA, J, J+JB-1, IPIV, 1 )

  IF( J+JB.LE.N ) THEN

    Apply interchanges to columns J+JB:N.

    CALL DLASWP( N-J-JB+1, A( 1, J+JB ), LDA, J, J+JB-1,
  $           IPIV, 1 )

    Compute block row of U.

    CALL DTRSM( 'Left', 'Lower', 'No transpose', 'Unit',
  $           JB, N-J-JB+1, ONE, A( J, J ), LDA,
  $           A( J, J+JB ), LDA )
  $
  IF( J+JB.LE.M ) THEN

    Update trailing submatrix.

    CALL DGEMM( 'No transpose', 'No transpose',
  $           M-J-JB+1, N-J-JB+1, JB, -ONE,
  $           A( J+JB, J ), LDA, A( J, J+JB ), LDA,
  $           ONE, A( J+JB, J+JB ), LDA )

  END IF
END IF
20 CONTINUE

```

PARALLEL LU FACTORIZATION CODE

```

DO 10 J = JA, JA+MIN(M,N)-1, DESCA( 5 )
  JB = MIN( MIN(M,N)-J+JA, DESCA( 5 ) )
  I = IA + J - JA

  Factor diagonal and subdiagonal blocks and test for exact
  singularity.

  CALL PDGETF2( M-J+JA, JB, A, I, J, DESCA, IPIV, IINFO )

  Adjust INFO and the pivot indices.

  IF( INFO.EQ.0 .AND. IINFO.GT.0 )
  $   INFO = IINFO + J - JA

  Apply interchanges to columns JA:J-JA.

  CALL PDLASWP( 'Forward', 'Rows', J-JA, A, IA, JA, DESCA,
  $           J, J+JB-1, IPIV )

  IF( J-JA+JB+1.LE.N ) THEN

    Apply interchanges to columns J+JB:JA+N-1.

    CALL PDLASWP( 'Forward', 'Rows', N-J-JB+JA, A, IA,
  $           J+JB, DESCA, J, J+JB-1, IPIV )

    Compute block row of U.

    CALL PDTRSM( 'Left', 'Lower', 'No transpose', 'Unit',
  $           JB, N-J-JB+JA, ONE, A, I, J, DESCA, A, I,
  $           J+JB, DESCA )
  $
  IF( J-JA+JB+1.LE.M ) THEN

    Update trailing submatrix.

    CALL PDGEMM( 'No transpose', 'No transpose',
  $           M-J-JB+JA, N-J-JB+JA, JB, -ONE, A,
  $           I+JB, J, DESCA, A, I, J+JB, DESCA,
  $           ONE, A, I+JB, J+JB, DESCA )

  END IF
END IF
10 CONTINUE

```

Conclusion

- Version multi-cores : PLASMA
- Version Out-of-Core
- Utilisé pour comparer les supercalculateurs (linpack pour top500)
- Top 1 : 1.3 Pflops (IBM) 122K cores (1 Pflops soutenu)
- PaStiX sur TERA1 : 25% puissance crête sur 1024 processeurs (en creux!)