Tutoriel : utilisation de LAPACK et des BLAS

Camille Coti

22 septembre 2016

1 Introduction

1.1 Un peu d'histoire

Les premières routines d'algèbre linéaire, les BLAS dites de niveau 1 (voir section 1.2), sont sorties en 1979, à peu près à la même période que le Cray 1. Leur but était de fournir une interface standardisée donnant accès à des implémentations à hautes performances, afin d'améliorer la portabilité des applications de calcul entre les supercalculateurs de l'époque, mais aussi de benchmarker les supercalculateurs.

C'est dans ce but qu'a été initié le projet LINPACK en 1974 : afin de comparer les performances des supercalculateurs, on leur fait exécuter à tous un programme qui résout un système dense d'équations linéaires en utilisant une factorisation LU avec pivotage partiel. Le premier rapport de benchmark est sorti en 1977 : la machine la plus rapide était un Cray 1 du NCAR (National Center for Atmospheric Research).

Les autres routines des BLAS ont été développées afin de fournir des opérations matricielles : matrice-vecteur ou matrice-matrice.

1.2 L'écosystème LAPACK

Les bibliothèques d'algèbre linéaire numérique de la famille de LAPACK dépendent entre elles via des relations représentées par la figure 1.

Les routines de base sont les BLAS (Basic Linear Algebra Subroutines). Leur interface est devenue un standard *de facto* Il en existe des implémentations performantes, spécialisées (liées au matériel) ou généralistes.

Les BLAS sont rangées dans 3 catégories : les `niveaux`

- *Niveau 1* : opérations *vectorielles* de type $y = \alpha x + y$, produit scalaire, produit vectoriel, norme... Les BLAS 1 sont sorties en 1979.
- *Niveau 2* : opérations *matrice-vecteur* de type $y = \alpha Ax + \beta y$, résolution de $Tx = y$ avec T triangulaire... Les BLAS 2 ont été développées en 1984-1986 et comptent 25 opérations.
- *Niveau 3* : opérations *matrice-matrice* de type $C = \alpha AB + \beta C$, résolution de $C = \alpha T^{-1}C$... Les BLAS 3 ont été développées en 1987-1988 et comptent 9 opérations.

Le "niveau" des BLAS vient de leur complexité : les BLAS de niveau 1 sont en $O(N)$, les BLAS de niveau 2 sont en $O(N^2)$ et les BLAS de niveau 3 sont en $O(N^3)$. Au total, les BLAS comptent 142 routines.

LAPACK fournit des routines de plus haut niveau : résolution de systèmes d'équations linéaires, moindres carrés, valeurs propres et décomposition en valeurs singulières. On y trouve aussi des routines de factorisations matricielles et d'estimation des nombres de conditionnement.

LAPACK utilise les routines des BLAS. En particulier, les routines de LAPACK font appel à des opérations matricielles des BLAS 3 pour effectuer des opérations par bloc, tenant dans le cache des machines. La première version est sortie en 1992.

LAPACK a rendu obsolète LINPACK (la bibliothèque, pas le benchmark) avec l'évolution des architectures matérielles. LINPACK se base sur des BLAS 1 : elle est donc adaptée aux architectures vectorielles. LAPACK se base elle sur des BLAS 3 : en permettant des opérations par blocs qui tiennent dans le cache, elle est adaptée aux machines à mémoire hiérarchique.

ScaLAPACK est une implémentation distribuée des routines de LAPACK. Elle se base sur LAPACK et les BLAS, ainsi que sur des routines de communications les BLACS) et des BLAS parallèles (les PBLAS).

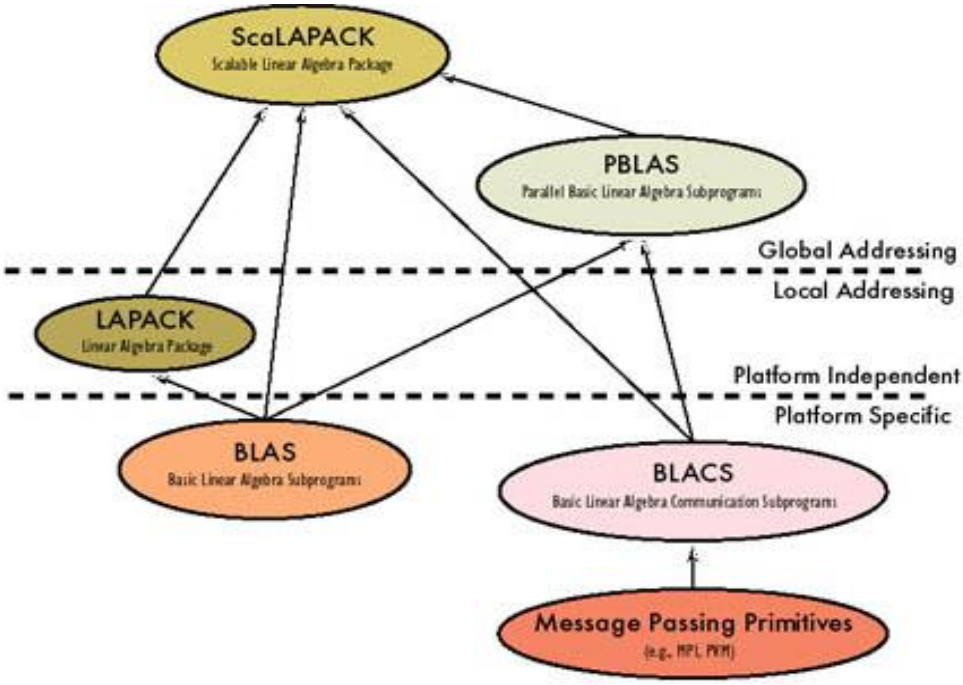

FIGURE 1 – Dépendances entre les bibliothèques d'algèbre linéaire numérique.

Il existe désormais des bibliothèques fournissant des implémentations spécialisée, comme PLASMA (Parallel Linear Algebra for Shared Memory Architectures) pour les architectures à mémoire partagée ou MAGMA pour les GPU.

1.3 Implémentations et installation

Les BLAS et LAPACK s'installent séparément. LAPACK faisant appel aux BLAS pour ses calculs de bas niveau, choisir des BLAS performantes est crucial. Les BLAS de référence, implémentées en Fortran et disponibles sur Netlib, ne sont là que pour servir de référence, comme leur nom l'indique. On a généralement le choix entre des BLAS fournies avec la machine (dans la bibliothèque MKL pour Intel, ACML pour AMD) ou des BLAS libres, comme l'excellent GotoBLAS.

Les bibliothèques MKL et ACML fournissent également les routines de LAPACK. Dans le cas où elles ne seraient pas disponibles, on peut l'installer la version disponible sur Netlib.

Il est également nécessaire de connaître l'existence d'ATLAS (Automatically Tuned Linear Algebra Software). Il s'agit d'un ensemble de routines fournissant les BLAS et certaines routines de LAPACK qui sont optimisées à la compilation : le système de compilation détermine de façon empirique les meilleurs paramètres à utiliser pour ses opérations et pour un grand ensemble de tailles de matrices, et ces paramètres sont ensuite ceux utilisés par la bibliothèque.

2 Conventions de nommage

Le nommage des routines suit toujours une convention qui permet de déterminer ce que fait la matrice et sur quels types de données.

— Lettre 1 : précision (S = simple, D = double, C = complexe, Z = complexe double précision)
— Lettres 2 et 3 : type de matrices (GE = générale, OR = orthogonale...)
— Lettes 4 à 6 : opération et algorithme (QRF, LU...)

BD	bidiagonal	DI	diagonal
GB	general band	GE	general [1]
GG	general matrices, generalized problem [2]	GT	general tridiagonal
HB	(complex) Hermitian band	HE	(complex) Hermitian
HG	upper Hessenberg matrix, generalized problem [3]	HP	(complex) Hermitian, packed storage
HS	upper Hessenberg	OP	(real) orthogonal, packed storage
OR	(real) orthogonal	PB	symmetric or Hermitian positive definite band
PO	symmetric or Hermitian positive definite	PP	symmetric or Hermitian positive definite, packed storage
PT	symmetric or Hermitian positive definite tridiagonal	SB	(real) symmetric band
SP	symmetric, packed storage	ST	(real) symmetric tridiagonal
SY	symmetric	TB	triangular band
TG	triangular matrices, generalized problem [4]	TP	triangular, packed storage
TR	triangular (or in some cases quasi-triangular)	TZ	trapezoidal
UN	(complex) unitary	UP	(complex) unitary, packed storage

2.1 Types de matrices

2.2 Opérations disponibles

Les opérations disponibles dans LAPACK se retrouvent grâce à leur nom en anglais :
— SV : Linear system of equations
— LLS : Linear least squares problems
— LSE : Linear equality-constrained least squares problem
— GLM : General linear model problem
— SEP : Symmetric eigenproblems
— NEP : Nonsymmetric eigenproblems
— SVD : Singular value decomposition
— GSEP : Generalized symmetric definite eigenproblems
— GNEP : Generalized nonsymmetric eigenproblems
— GSVD (QSVD) : Generalized (or quotient) singular value decomposition

2.3 Exemples

Par exemple, DGELUB désigne une factorisations LU sur une matrice générale en double précision. ZPOTRF désigne une factorisation de Cholesky sur une matrice définie positive (PO) en complexes double précision.

On retrouve la même convention de nommage dans les routines de toutes les bibliothèques de cet écosystème. Par exemple, la routine DGEMM (BLAS de niveau 3) désigne une multiplication matrice-matrice (MM) pour matrices générales (GE) en double précision (D).

3 Utilisation de l'interface Fortran

Même la plupart des BLAS sont écrits en C en-dehors des BLAS de référence, LAPACK est écrit en Fortran. On peut appeler directement ses fonctions depuis un code en C ou C++.

3.1 Interface

Certains fichiers d'en-têtes sont disponibles pour vous fournir les prototypes des fonctions (`lapack.h`). Dans le cas où vous n'en utiliseriez pas, vous devez définir quelque part dans votre code l'en-tête des fonctions de LAPACK ou des BLAS que vous allez appeler.

Prenons l'interface F77 de la fonction `dgetrf` [5].

```
1   SUBROUTINE DGETRF( M, N, A, LDA, IPIV, INFO )
```

En lisant les commentaires dans le fichier qui la définit, on lit que :
— M est un entier
— N est un entier
— A est un tableau en double précision, de taille (LDA,N)
— LDA est un entier
— IPIV est un tableau d'entiers, de dimension (min(M,N))
— INFO est un entier

En Fortran, les paramètres sont passés par référence. L'interface C utilise donc des pointeurs pour tous ces paramètres. On obtient alors l'interface C :

```
1 void dgetrf( int *m, int *n, double *a, int *lda, int *ipiv, int *info );
```

Certains compilateurs ajoutent un _ à la fin des noms de fonctions. Il est donc possible que vous ayez besoin d'en ajouter un à la fin du nom de la fonction, ou plus simplement de définir une instruction de pré-processing :

```
1   #define dgetrf dgetrf_
```

On peut alors faire quelques remarques sur les données manipulées dans LAPACK et les BLAS, qui sont héritées du Fortran.

Tout d'abord, les tableaux sont tous des tableaux à une dimension. Les données des tableaux à deux dimensions sont linéarisées en ordre *column major*.

C'est pour cela que l'on utilise la *leading dimension*. Par définition, elle correspond à l'espace entre deux colonnes consécutives. Dans le cas général, il s'agit du nombre de lignes : si vous manipulez une matrice 100x100, le nombre de lignes sera 100 et sa leading dimension sera 100. Cependant, si vous ne manipulez qu'une sous-matrice de la matrice d'origine, deux lignes consécutives seront séparées en mémoire par une plus grande distance que le nombre de colonnes de la sous-matrice. Si l'on prend une sous-matrice 10x10 de notre matrice 100x100, deux lignes consécutives seront séparées par 100 cases alors que le nombre de colonnes est de 10.

Prenons maintenant un autre exemple : la multiplication matrice-matrice avec `dgemm`. Cette fonction réalise l'opération $C = \alpha AB + \beta C$. Les matrices A et B peuvent être transposées, et α et β sont des constantes. Son interface C est la suivante :

```
1   void dgemm( char *transa, char *transb, int *m, int *n, int *k,
2               double *alpha, double *a, int *lda, double *b, int *ldb,
3               double *beta, double *c, int *ldc );
```

On passe ici plusieurs paramètres constants :
— `transa` : la matrice A doit-elle être transposée
— `transb` : la matrice B doit-elle être transposée
— `alpha` : la constante multiplicative α
— `beta` : la constante multiplicative β

Cependant, comme les autres paramètres, il faut les passer par pointeurs. On doit alors définir des variables qui les contiennent et dont les adresses seront passées comme paramètres, par exemple :

```
1   char Transpose='T';
2   double ONE = 1.0;
```

3.2 Compilation

Le code que vous venez d'écrire se compile avec un compilateur correspondant au langage utilisé. Cependant, l'édition des liens nécessite une attention particulière. En effet, vous assez lier un code écrit en C ou C++ avec une bibliothèque Fortran.

5. http://www.netlib.org/lapack/explore-3.1.1-html/dgetrf.f.html

Si votre code est en C vous avez deux possibilités : effectuer l'édition des liens avec votre compilateur Fortran, ou avec votre compilateur C en ajoutant la bibliothèque Fortran dans les liens. Le compilateur Fortran effectue l'édition des liens avec la bibliothèque Fortran, tandis que le compilateur C ne le fait pas et il faut donc l'ajouter explicitement.

Par exemple, en utilisant les compilateurs GNU (gcc et gfortran), la bibliothèque Fortran s'appelle libgfortran. Vous avez donc deux possibilités :

```
1  gcc -O3 -c moncode.c
2  gfortran -o moncode moncode.o
```

Ou en effectuant les deux étapes avec GCC :

```
1  gcc -O3 -c moncode.c
2  gcc -o moncode moncode.o -lgfortran
```

Si votre code est en C++ vous devez effectuer l'édition des liens avec la bibliothèque Fortran et la bibliothèque C++. vous pouvez donc soit le faire avec un compilateur Fortran et ajouter la bibliothèque C++, ou le faire avec un compilateur C++ et ajouter la bibliothèque Fortran.

Toujours en utilisant les compilateurs GNU, la bibliothèque C++ s'appelle libstdc++. Vous avez ici encore deux possibilités :

```
1  g++ -O3 -c moncode.cxx
2  gfortran -o moncode moncode.o -lstdc++
```

Ou en effectuant les deux étapes avec G++ :

```
1  g++ -O3 -c moncode.cxx
2  g++ -o moncode moncode.o -lgfortran
```

3.3 Exemple

Petit exemple : multiplication de deux matrices avec `dgemm`. Les deux matrices sont remplies de valeurs aléatoires avec `dlarnv`.

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  #define dgemm dgemm_
5  #define dlarnv dlarnv_
6
7  #define MATSIZE 256
8
9  static int IONE = 1;
10 static double c__1 = 1.0;
11 char NoTranspose = 'N';
12 int ISEED[4] = {0,0,0,1};
13
14 int main (int argc, char **argv){
15
16     double *A, *B, *C;
17     int N = MATSIZE;
18     int NN = N*N;
19
20     /* Allocate data */
21     A    = (double *)malloc(N*N*sizeof(double));
22     B    = (double *)malloc(N*N*sizeof(double));
23     C    = (double *)malloc(N*N*sizeof(double));
```

```
    /* Fill the matrices with random values */
    dlarnv( &IONE, ISEED, &NN, A );
    dlarnv( &IONE, ISEED, &NN, B );
    dlarnv( &IONE, ISEED, &NN, C );
    /* Perform the operation: C = 1.0*A*B + 1.0*C */
    dgemm(&NoTranspose,&NoTranspose,&N,&N,&N,&c__1,A,&N,B,&N,&c__1,C,&N);

    free(A);
    free(B);
    free(C);
    return EXIT_SUCCESS;
}
```

Pour les compiler, on peut utiliser le Makefile suivant. Il utilise `gcc` pour la génération du fichier objet et `gfortran` pour l'édition des liens, et les BLAS sont dans la bibliothèque `libopenblas`.

```
CC = gcc
CCOPT = -O3
LD = gfortran
LDLIBS = -lopenblas

exempledgemm: exempledgemm.o
	$(LD) -o exempledgemm exempledgemm.o $(LDLIBS)

exempledgemm.o: exempledgemm.c
	$(CC) $(CCOPT) -c exempledgemm.c

clean:
	@rm -f exempledgemm exempledgemm.o

.PHONY: clean
```

4 Utilisation de lapacke