Université Pierre
et Marie Curie

M2 AN&EDP

Projet de "Calcul haute performance, algorithmes parallèles d'algèbre linéaire à grande échelle, stabilité numérique"

Parallélisation d'un algorithme probabiliste de décomposition Low-rank

Auteur :
Olivier Tissot

Responsable du cours :
Mme. Laura Grigori

21 mai 2014

Table des matières

1 Etude préliminaire

1.1 Description générale de l'algorithme

On commence par une définition :

Définition 1. *Soient une matrice A de taille $m \times n$ et un rang réduit l (souvent choisi tel que $l \ll min(m,n)$). Une factorisation Low-Rank de A est la donnée de deux matrices B et C de tailles respectives $m \times l$ et $l \times n$ et telles que $||A - BC||$ soit minimale.*

Remarque 1. *Puisqu'on est dans un espace de dimension finie, toutes les normes sont équivalentes. Pour se fixer les idées, on peut imaginer la norme de Frobenius ou une norme subordonnée.*

L'algorithme étudié est le suivant [1] :

Algorithme 1: Décomposition SVD de A (version sans parallélisme)

Input : A matrice $m \times n$, l le rang de l'approximation Low-Rank

Factorisation Low-Rank probabiliste : $A \approx BC$
Calcul de G matrice $n \times l$ formée de tirages aléatoires, indépendants de loi $\mathcal{N}(0,1)$
$H \leftarrow AG$
Calcul de Q matrice $m \times l$ dont les vecteurs forment une base orthonormale de l'image de H
$B \leftarrow Q$
$C \leftarrow Q^\top A$

Décomposition SVD de la factorisation Low-Rank obtenue
Calcul de la SVD de C : U, Σ, V

Output : QU matrice unitaire $m \times m$, Σ matrice diagonale $m \times n$, $V^\top$ matrice unitaire $n \times n$

Dans un premier temps, on réalise une factorisation Low-Rank de A. Pour ce faire, on commence par calculer une image approchée de A : $H = AQ$. Et ensuite, on projette orthonormalement les colonnes de A sur cette image approchée. De sorte que $A \approx QQ^\top A$. Dans [WC], il est démontré qu'en fait cette procédure fournit une bonne approximation : si les valeurs singulières de A sont toutes égales à 1 on retrouve une borne optimale.

1. Il s'agit de l'algorithme 2 présenté dans [WC], disponible sur la page d'Emmanuel Candès.

Remarque 2. *Plus précisément, on a le résultat suivant :*

$$\mathbb{E}\left[||(I - QQ^\top)A||\right] \leqslant \left(1 + \sqrt{\frac{k}{p-1}}\right)\sigma_{k+1} + e\frac{\sqrt{k+p}}{p}\sqrt{\sum_{i>k}\sigma_i^2}$$

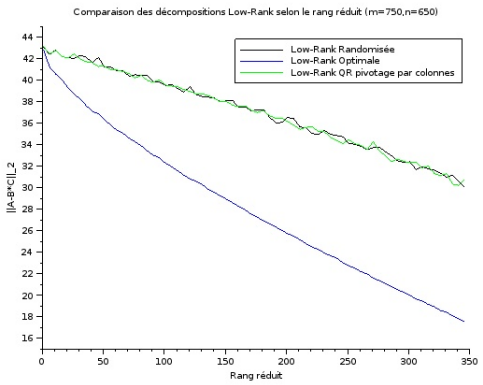
Avec : $l = k + p$ et $p > 0$, et σ_i les valeurs singulières de A.

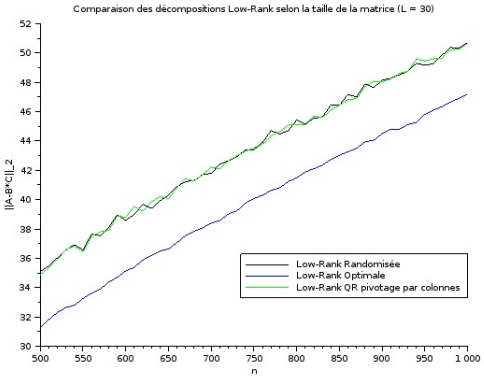
Et ensuite, on applique une factorisation SVD à la matrice réduite. Comme on sait que Q est orthonormale, il suffit en fait de décomposer $C = Q^\top A$ puis de multiplier la matrice U par Q (par la gauche bien sûr).

Remarque 3. *On peut remplacer l'étape de factorisation SVD par une factorisation QR. Si on note $\hat{Q}\hat{R}$ la décomposition QR de C alors $Q\hat{Q}$ et $\hat{R}$ est une décomposition QR approchée de A.*

1.2 Comportement numérique

Pour étudier le comportement numérique de l'algorithme, on a utilisé Scilab [2]. Pour les premiers tests, la matrice A est une matrice formées de tirages aléatoires, indépendants de loi $\mathcal{U}([-1, 1])$. On a comparé 3 factorisations Low-Rank différentes : celle présentée précédemment [3], la factorisation SVD tronquée et la factorisation QR avec pivotage par colonnes.





On s'est concentré sur la partie décomposition Low-Rank car on peut considérer que l'erreur commise lors d'une décomposition SVD ou QR est négligeable [4]. L'approche randomisée et la factorisation QR fournissent des approximations très proches l'une de l'autre. Ce n'est pas vraiment étonnant car les idées derrières les 2 algorithmes sont finalement très proches. La

2. cf. Annexes pour le code source du script Scilab.
3. cf. Algorithme 1.
4. De l'ordre de 10^{-16} contre 10^1.

factorisation issue de la décomposition SVD est la plus précise. On retrouve un résultat théorique puisqu'on peut démontrer que c'est la décomposition optimale, mais c'est aussi la plus coûteuse [5]. On remarque néanmoins que lorsque $l \ll min(n, m)$ l'écart est moins important. En particulier, si n et m sont grands les factorisations QR et randomisée deviennent très proches de la décomposition issue de la SVD.

5. En pratique, on n'arrive pas à calculer des décompositions SVD pour de grandes matrices car la complexité de l'algorithme est en $O(n^3)$

2 Parallélisation de l'algorithme

Dans un souci de simplification, on laisse de côté la phase de génération aléatoire de G. Cette étape est très parallélisable car toutes les réalisations à l'intérieur de G sont indépendantes et il suffit juste de savoir générer une réalisation d'une variable aléatoire de loi normale centrée réduite [6].

2.1 Produit Matrice-Matrice

Une fois la matrice G calculée, il nous faut calculer le produit matriciel $A \times G$. Dans un premier temps, on doit choisir une façon de répartir les données entre les processeurs. Afin de rester le plus général possible on choisit une répartition en blocs cyclique [7]. On obtient donc l'algorithme suivant :

Algorithme 2: Produit $A \times G$ parallélisé

Input : A matrice $m \times n$, H matrice $n \times l$, $p = p_r \times p_c$ grille de processeurs, b taille des blocs

for $k = 1$ **to** $m/b - 1$ **step** b **do**
  /* *Generate* G *of dimension* $b \times b$ */
  `matgen`(G, b, b)
  for $i = 1$ **to** p_r **do**
    `MPI_Gather`$(G(k : k+b; i), G_r, Comm\_Col)$
  end
  for $j = 1$ **to** p_c **do**
    `MPI_Gather`$(A(j; k : k+b), A_c, Comm\_Row)$
  end
  $H \leftarrow H + A_c G_r$
end

Output : H matrice $(m \times l)$

k_γ	k_β	k_α
$O(\frac{nml}{p})$	$O(\frac{mn}{\min(p_r,p_c)} log(\max(p_r,p_c)))$	$O(\frac{m}{b} log(\max(p_r,p_c)))$

TABLE 1 – Coût de l'algorithme dans le système (γ, β, α)

Remarque 4. *Cet algorithme appelé* SUMMA *celui utilisé dans la bibliothèque* PBLAS [8].

6. Par exemple, on peut utiliser la méthode de Box-Muller.
7. Les autres répartitions peuvent se déduire de celle-ci.
8. *Parallel BLAS*

2.2 Décomposition QR

Pour calculer une base orthonormale de H et pour calculer la décomposition SVD de C, on aura besoin de calculer une décomposition QR. Etant donnée une matrice A de taille $n \times m$, on va chercher à expliciter Q de taille $m \times m$ et R de taille $m \times n$ telles que : $A = QR$, $Q^\top Q = I$ et R est triangulaire supérieure.

2.2.1 Méthode de Householder

Cette méthode est très utilisée en pratique car elle est beaucoup plus stable numériquement qu'une procédure de Gram-Schmidt. Elle utilise de manière centrale les matrices de la forme $I - YTY^\top$ qu'on appelle matrices de Householder et qui possèdent des propriétés géométriques intéressantes.

Algorithme 3: Factorisation QR parallélisée

Input : A matrice $m \times l$, $p = p_r \times p_c$ grille de processeurs, $b = b_r \times b_c$ taille des blocs

for $ib = 1$ **to** $n - 1$ **step** b **do**

- ** Compute panel factorization **
- $\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} = Q_1 R_{11} = (I - YTY^\top) R_{11}$
- ** Broadcast along the rows of Y and T **
- `MPI_Gather`$(Y, len\_glob\_Y, Comm\_Row)$
- `MPI_Gather`$(T, len\_glob\_T, Comm\_Row)$
- ** Apply $Q_1^\top$ on the trailing matrix **
- $W_1 = Y_1^\top \begin{pmatrix} A_{12}^{loc} \\ A_{22}^{loc} \end{pmatrix}$
- ** If the process owns block row ib **
- **if** $iA == ib$ **then**
  - $W = Y^\top \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix}$
  - $W' = T^\top W$
- **end**
- `MPI_Gather`$(W', len\_glob\_W', Comm\_Col)$
- $\begin{pmatrix} R_{12}^{ib} \\ R_{22}^{ib} \end{pmatrix} = \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} - \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} * W'$

end

Output : Q orthogonale $m \times m$ et R triangulaire supérieure $m \times l$

Remarque 5. *Dans* ScaLAPACK*, la matrice Q n'est pas explicitée et il faut utiliser la fonction* `PDORMQR` *pour pouvoir calculer $Q^\top A$.*

k_γ	k_β	k_α
$O(\frac{2ml^2}{p} - \frac{2l^3}{3p}log(p))$	$O(\frac{2ml}{p_r}log(\max(p_r, p_c)))$	$O(2l\log(\max(p_r, p_c)))$

TABLE 2 – Coût de l'algorithme dans le système (γ, β, α)

Remarque 6. *Si le rang de A est plus petit que n alors rien ne nous garantit que $Im(A) = Im(Q)$. Néanmoins, si on suppose que $l \ll min(m, n)$, on peut raisonnablement en déduire que le rang de A est égal à l. Ceci justifie l'utilisation de cet algorithme pour calculer la matrice Q dans l'Algorithme 1.*

2.2.2 Pivotage par colonnes

En raison de la remarque précédente, on aura besoin de modifier l'algorithme précédent si on veut utiliser une décomposition QR pour calculer une décomposition SVD de C. En effet, puisque C est de dimension $l \times n$ alors nécessairement le rang de C est inférieur à n.

Une première approche est donc d'utiliser une étape de pivotage par colonne dans l'algorithme précédent. En effet, on remarque que si le rang de A vaut k alors à l'étape $k+1$ on aura $\max_{j=k,\dots,n} ||r_j||_2 < \epsilon_{machine}$ avec r_j les colonnes de R_{22}^k. Donc à l'étape k, on calcule $||r_m||_2 = \max_{j=k,\dots,n} ||r_j||_2$ et si $||r_k||_2 > \epsilon_{machine}$ alors on permute r_k et r_m et on continue l'algorithme. Cette étape va entraîner un surcoût dans les communications pour pouvoir calculer $||r_m||_2$ mais le nombre de flops reste le même.

k_γ	k_β	k_α
$O(\frac{2ml^2}{p} - \frac{2l^3}{3p}log(p))$	$O(\frac{2ml}{p_r}log(\max(p_r, p_c)))$	$O(3l\log(\max(p_r, p_c)))$

TABLE 3 – Coût de l'algorithme dans le système (γ, β, α)

2.2.3 Pivotage par tournoi

Afin de minimiser les communications, on peut aussi utiliser une variante de l'approche précédente présentée dans [DGGX] : la factorisation QR avec

pivotage des colonnes par tournoi ou $CARRQR$. L'idée est d'écrire :

$$\begin{aligned}\begin{pmatrix}A_1\\A_2\\A_3\\A_4\end{pmatrix} &= \begin{pmatrix}Q_1^1R_1^1\\Q_2^1R_2^1\\Q_3^1R_3^1\\Q_4^1R_4^1\end{pmatrix} = \begin{pmatrix}Q_1^1 & & & \\ & Q_2^1 & & \\ & & Q_3^1 & \\ & & & Q_4^1\end{pmatrix}\begin{pmatrix}R_1^1\\R_2^1\\R_3^1\\R_4^1\end{pmatrix}\\ &= \widetilde{Q^1}\begin{pmatrix}Q_1^2R_1^2\\Q_2^2R_2^2\end{pmatrix} = \widetilde{Q^1}\begin{pmatrix}Q_1^1 & \\ & Q_2^2\end{pmatrix}\begin{pmatrix}R_1^2\\R_2^2\end{pmatrix}\\ &= \widetilde{Q^1}\widetilde{Q^2}\widetilde{Q^3}R^3\end{aligned}$$

Où à chaque étape on effectue une factorisation $RRQR$[9] comme la factorisation QR avec pivotage par colonnes. Cet algorithme permet de minimiser les communications au prix d'un léger surcoût sur le nombre de flops. En pratique, comme ce sont les communications qui sont les plus coûteuses, il est plus intéressant que l'algorithme précédent et devrait offrir de meilleures performances.

k_γ	k_β	k_α
$O(\frac{2ml^2}{p} + \frac{2l^3}{3p}log(p))$	$O(\frac{2ml}{p_r}log(\max(p_r,p_c)))$	$O(\frac{l}{\min(p_r,p_c)}log(\max(p_r,p_c)))$

TABLE 4 – Coût de l'algorithme dans le système (γ, β, α)

2.3 Scalabilité de l'algorithme

$log_2(n)$ \ $log_2(p)$	1	2	3	4	5	6	7	8	9
10	0.526	0.672	0.748	0.794	0.824	0.843	0.856	0.862	0.863
11	0.694	0.727	0.768	0.802	0.827	0.845	0.856	0.862	0.863
12	1.315	1.044	0.908	0.861	0.852	0.855	0.861	0.864	0.864
13	1.855	1.677	1.423	1.179	1.016	0.932	0.895	0.880	0.871
14	1.979	1.946	1.875	1.741	1.534	1.299	1.107	0.987	0.922
15	1.997	1.992	1.980	1.956	1.905	1.806	1.639	1.418	1.199
16	2.000	1.998	1.996	1.992	1.984	1.966	1.930	1.858	1.726
17	2.000	2.000	1.999	1.998	1.996	1.993	1.987	1.974	1.947
18	2.001	2.000	1.999	1.999	1.998	1.997	1.996	1.993	1.988
19	2.001	2.000	2.000	1.999	1.999	1.999	1.998	1.997	1.996

TABLE 5 – Scalabilité forte pour $(\gamma = 2.10^{-12}, \beta = 2.10^{-9}, \alpha = 10^{-5})$

9. Rank Revealing QR.

3 Implémentation informatique

3.1 Fonctionnalités du prototype

Le prototype est écrit en C et il utilise la biliothèque *ScaLAPACK*[10]. Cette bibliothèque étant initialement écrite en Fortran, on utilise des wrapers afin d'appeler en C des fonctions écrites en Fortran. L'utilisation de cette bibliothèque implique quelques restrictions :

- il faut avoir *ScaLAPACK* et ses dépendances [11] d'installées sur la machine
- A est une matrice pleine
- les processeurs seront distribués selon une grille

Actuellement, le prototype est limité aux fonctionnalités suivantes :

- étape de factorisation Low-Rank, i.e la première partie de l'Algorithme 1
- pas de pivotage par tournoi dans la factorisation QR : utilisation de la routine `PDGEQRF` de *ScaLAPACK*
- affichage du temps de calcul et de la norme Frobenius du résidu dans le fichier d'output

Un fichier d'input permet de donner des valeurs pour les paramètres suivants sans avoir à recompiler le programme :

- taille de A
- valeur de l
- dimension de la grille (attention elle doit être cohérente avec le nombre de processeurs donné à l'exécution)

La plupart des erreurs sont gérées en tenant compte du parallélisme afin de pouvoir débuguer plus facilement.

3.2 Premiers résultats

Les tests effectués pour le moment peuvent être qualifiés de tests de debugage du prototype. Ils ont tous été effectués sur un ordinateur portable donc le processeur est dual-core.

Dans un premier temps, on a choisi $m = n = l$, $p_r = p_c$ et $b = 1$, de cette façon on limite les erreurs liées à des dépassements de capacité. Dans cette configuration, on a testé le produit matrice-matrice et la décomposition en prenant la matrice identité de façon à vérifier que les arguments passés aux fonctions *ScaLAPACK* étaient corrects. Ensuite, on a essayé de prendre des matrices rectangulaires puis on a effectué les mêmes tests. Enfin, on a changé les taille des blocs et les dimensions de la grille de processeur. En particulier, on a testé qu'une distribution ligne cyclique fonctionnait.

10. Scalable LAPACK
11. Se reporter au site officiel pour avoir la liste complète.

Dans un second temps, on a choisi $n = l$ puis on a calulé $||A - BC||$. En effet, si $l = n$ on doit normalement avoir $A = BC$ exactement. Numériquement, on doit donc retrouver la précision machine pour des valeurs de m, n et l suffisamment petites [12], et ce, quelque soit la norme car elles sont équivalentes. Le prototype nous donne une erreur de l'ordre de $1e - 15$ pour la norme Frobenius et la norme 1 pour $m = 8$ et $n = l = 4$. On considère donc que ce test est réussi.

3.3 Perspectives

Bien qu'ils permettent de s'appuyer sur une base de développement saine, ces premiers tests ne sont pas suffisants pour valider entièrement le prototype et répondre au problème initial. Notamment, il serait intéressant d'étudier expérimentalement la scabilité de l'algorithme sur une machine massivement parallèle.

Il faudrait aussi compléter l'algorithme en calculant la factorisation SVD de C et effectuer des tests de validité du développement : robustesse, précision, comparaison avec les autres factorisations Low-Rank.

Ensuite, il serait intéressant d'implémenter quelques autres fonctionnalités au programme :

- pouvoir choisir A creuse
- pouvoir récupérer A à partir d'un fichier (au format *Matlab* par exemple)

Enfin, pour aller encore plus loin, on pourrait envisager d'écrire une version modifiée de `PDGEQRF` pour effectuer une décomposition QR avec pivotage par tournoi afin de tester la différence entre les 2 factorisations.

12. Sinon les erreurs d'arrondis peuvent se cumuler et l'erreur devient plus grande que la précision machine.

Conclusion

On a commencé par présenter un algorithme randomisé pour calculer une approximation Low-Rank d'une matrice de grande taille. L'une des applications est le calcul de la SVD sur l'approximation ainsi obtenue.

Ensuite, on a explicité la parallélisation du calcul de l'approximation Low-Rank. On a indiqué des estimations des différents coûts dans le système (γ, β, α), ainsi qu'une estimation des scalabilités forte faible sur une machine massivement parallèle en donnant des valeurs à ces paramètres.

Enfin, on a présenté un prototype de code qui implémente la décomposition Low-Rank randomisée en parallèle avec MPI. Ce prototype, basé sur des routines *ScaLAPACK*, a été testé sur quelques cas très simples et sur un ordinateur personnnel.

Références

[WC] R.Witten et E.Candès, *Randomized Algorithms for Low-Rank Matrix Factorizations : Sharp Performance Bound*, Août 2013.

[DGGX] J.Demmel, L.Grigori, M.Gu et H.Xiang, *Communication-avoiding Rank-revealing QR Decomposition*, 2013.

Annexes

Annexe A : Scripts Scilab

Listing 1 – Calcul des 3 décompositions Low_Rank et des résidus.

```
stacksize('max');
m = input(" m <- ");
n = input(" n <- ");
l = input(" l <- ");

Rrand = 0;
Ropti = 0;
Rqrpc = 0;

// Decomposition Low-Rank randomisee
A = grand(m, n, "uin",-1, 1);
G = grand(n, l, "nor", 0, 1);
H = A*G;
Q = orth(H);
C = Q'*A;

// Decomposition Low-Rank optimale
[U,S,V] = svd(A);
U1 = U(:,1:l);
S1 = S(1:l,1:l);
V1 = V(:,1:l)';

// Decomposition QR pivotage par colonne
[Qt,Rt] = qr(A);
Qtt = Qt(:,1:l);
Rtt = Rt(1:l,:);

Rrand = [Rrand norm(A-Q*C,2)];
Ropti = [Ropti norm(A-U1*S1*V1,2)];
Rqrpc = [Rqrpc norm(A-Qtt*Rtt)];

// Comparaison avec la decomposition Low-Rank optimale
printf("Residu pour la reduction randomisee :
    %g\n",norm(A-Q*C,2))
printf("Residu pour la reduction optimale :
    %g\n",norm(A-U1*S1*V1,2))

// Comparaison des decompositions QR
```

```
printf("Residu pour QR randomisee :
    %g\n",norm(A-Q*Qt*Rt,2))
printf("Residu pour QR optimale :
    %g\n",norm(A-Qtt*Rtt,2))
```

Listing 2 – Calcul de la scalabilité.

```
stacksize('max');
alpha = 10^-5;
beta = 20^-9;
gamma = 20^-12;
time = eye(10,10);
l = 200;
mval = 0;
pval = 0;
strong_sca = eye(10,6);

// Approximate time in seconds
for km=20:29
    m = 2^km;
    mval = [mval m];
    for kp=1:9
        p = 2^kp;
        pval = [pval p];
        pr = sqrt(p);
        b = m/pr;
        time(km-19,kp) = gamma*(2*m*b^2/p + 4*m^3*l/p
            - 2*l^3/3*log(pr)) + beta*(2*m^2/pr*log(pr)
            + (2*m*l-l^2)/pr) + alpha*(2*m/b*log(pr) +
            2*l*log(pr));
    end
end

weak_sca = zeros(6);
// Scalability
for i=1:10
    for j=1:6
        strong_sca(i,j) = time(i,j)/time(i,j+1);
    end
end
for j=1:6
    weak_sca(j) = time(j,j)/time(j+1,j+1);
end
```

```
// Saving data
//write("weak_scalability.dat",weak_sca);
write("Scilab/strong_scalability_up.dat",strong_sca);
x = 1:size(weak_sca,1);
//plot2d(x,size(strong_sca,2))
//plot2d(x,strong_sca(2,:));
//xtitle('Approximation de la scalabilite forte,
    N=16384');
```

Annexe B : Code source prototype

Listing 3 – Fichier principal

```
/* prototype.c ---
 *
 * Filename: prototype.c
 * Description:
 * Author: O.TISSOT
 * Created: ven. mai  9 14:48:36 2014 (+0200)
 * Version:
 * Last-Updated: sam. mai 10 20:37:30 2014 (+0200)
 *           By: Olivier
 *     Update #: 159
 * Compatibility:
 * ScaLAPACK and PBLAS (and their dependancies)
 */

/* Commentary:
 * A simple prototype of the algorithm studied in the
    project.
 * It just do the low-rank factorization part.
 * I used examples  written by Kelly McQuighan for the
 * Kobe-Brown summer school on high performance
    simulations.
 * They are available on her personnal page :
 * http://www.dam.brown.edu/people/kellym
 */


/* Code: */

#include <stdio.h>
#include <stdlib.h>
```

```c
#include <math.h>
#include <time.h>
#include <string.h>

#include "scalapack_headers.h"
#include "blacs_headers.h"
#include "pblas_headers.h"
#include "scalapack_tools_headers.h"
#include "pspblasinfo.h"
#include "psmatgen.h"
#include "stringTools.h"
#include "pblasIOtools.h"

static int imax( int a, int b ){
  if (a>b) return(a); else return(b);
}

static int imin( int a, int b ){
  if (a<b) return(a); else return(b);
}

//------MAIN----------------------
int main(int argc, char *argv[])
{
  int i_am, k, m, n, num_proc_col, num_procs,
     num_proc_row, block_size, sys_param[6], status;
  int descA[9], descB[9], descC[9], info, maxmp,
     maxkp, maxmkn, minmpn;
  int kp, kq, mp, my_col, my_row, nq, contxt, izero =
     0, ione = 1, len_mat_name, lwork, ltau;
  int iaseed = 100, ibseed = 200, outfile_name_len;
  FILE* outfile=NULL;
  char folder[] = "data";
  const char * infile_name = "data/params.dat";
  char outfile_name[100], usr_info[100];
  double *A=NULL, *B=NULL, *C=NULL, *A0=NULL,
     *C0=NULL, *tau=NULL, *work=NULL;
  double done = 1.0e0, dmone = -1.0e0, resid = 0.0e0,
     eps = 1e-12, wall_time = 0.0e0;

  //*****************************************************
  //                  INITIALIZE BLACS
  //*****************************************************
```

```
  Cblacs_pinfo( &i_am, &num_procs );
  // read data from file BLAS.dat. Exit program if
     file not formatted properly
  pspblasinfo( &m, &n, &k, &block_size, &num_proc_row,
     &num_proc_col, sys_param, &i_am, &num_procs,
     folder, infile_name, outfile_name, usr_info,
     &outfile_name_len, &status );
  if ( status < 0 ) {
    return EXIT_FAILURE;
  }

  if (i_am == 0){
    outfile = fopen( "data/results.out", "w" );

    // output data
    fprintf ( outfile, "This is a prototype of the
       randomized low-rank factorization described in
       :\n" );
    fprintf ( outfile, " ''Randomized algorithms for
       for low-rank matrix factorizations : sharp
       performance bounds'', E. Candes & R. Witten,
       2013\n" );
    fprintf ( outfile, "It is available on E. Candes's
       web page.\n" );
    fprintf ( outfile, "It is written in parallel
       thanks to ScaLAPACK by Olivier Tissot. It is a
       part of a project for Laura Grigori's class
       about parallel computing given in 2nd semester
       of 2013-2014 of Master 2 AN&EDP of UPMC. A
       little report explains the methodology of the
       work. If you want it, please send me an e-mail
       : oli.tissot@gmail.com.\n" );
    fprintf ( outfile, "The code is inspired from
       examples written by Kelly McQuighan for a
       summer school on high performance simulations.
       They are availables on her webpage.\n" );
    fprintf ( outfile,
       "*****************************************************************
    fprintf ( outfile, "An explanation of the
       input/output parameters is as follows :\n" );
    fprintf ( outfile, "m\t\t: The number of rows in
       the matrices A and C.\t\t\tm\t\t= %d\n", m );
    fprintf ( outfile, "n\t\t: The number of columns
       in the matrices B and C.\t\tn\t\t= %d\n", n );
```

```
    fprintf ( outfile, "k\t\t: The number of rows of B
       and the number of columns of A.\tk\t\t= %d\n",
       k );
    fprintf ( outfile, "block_size\t: The size of the
       square blocks the matrices A, B, and C
       are\n\t\t  split
       into.\t\t\t\t\t\tblock_size\t= %d\n",
       block_size );
    fprintf ( outfile, "num_proc_row\t: The number of
       process rows.\t\t\t\tnum_proc_row\t= %d\n",
       num_proc_row );
    fprintf ( outfile, "num_proc_col\t: The number of
       process columns.\t\t\tnum_proc_col\t= %d\n",
       num_proc_col );
    fprintf ( outfile,
       "*****************************************************************
    fprintf ( outfile, "\nProgram progress:\n" );
  }

  if (num_proc_row*num_proc_col > num_procs){
    if (i_am==0) {
      fprintf ( stdout,
         "\n*****************************************************************
      fprintf ( stdout, "ERROR : we do not have enough
         processes available to make a nprow x npcol
         process grid\n\n" );
      fprintf ( stdout,
         "*****************************************************************
    }
    return EXIT_FAILURE;
  }

  Cblacs_get( -1, 0, &contxt );
  Cblacs_gridinit( &contxt, "Row", num_proc_row,
     num_proc_col );
  Cblacs_gridinfo( contxt, &num_proc_row,
     &num_proc_col, &my_row, &my_col );

  // only continue if I'm on the grid
  if ( ( my_row < num_proc_row ) & ( my_col <
     num_proc_col ) ) {

   // determine local matrix dimensions on each
      processor
```

```
// usage: A_loc = mp x kq, B_loc = kp x nq, C_loc =
   mp x nq
mp = numroc_( &m, &block_size, &my_row, &izero,
   &num_proc_row );
kp = numroc_( &k, &block_size, &my_row, &izero,
   &num_proc_row );
kq = numroc_( &k, &block_size, &my_col, &izero,
   &num_proc_col );
nq = numroc_( &n, &block_size, &my_col, &izero,
   &num_proc_col );
maxmp = imax(1,mp);
maxkp = imax(1,kp);
maxmkn = imax(imax(m,k),n);
minmpn = imin(n, m);
lwork = maxmkn*maxmkn;
ltau = minmpn;

// initialize descriptors for the matrices A, B, C
   and R
descinit_( descA, &m, &k, &block_size,
   &block_size, &izero, &izero, &contxt, &maxmp,
   &info );
descinit_( descB, &k, &n, &block_size,
   &block_size, &izero, &izero, &contxt, &maxkp,
   &info );
descinit_( descC, &m, &n, &block_size,
   &block_size, &izero, &izero, &contxt, &maxmp,
   &info );

if (i_am==0) fprintf ( outfile, "BLACS functions
   numroc and descinit completed successfully.\n");

//*******************************************************
//                  INITIALIZE MATRICES
//*******************************************************
// allocate memory for the local part of matrices
   A, B, C and R
A    = (double*) malloc(maxmp*kq*sizeof(double));
A0   = (double*) malloc(maxmp*kq*sizeof(double));
B    = (double*) malloc(maxkp*nq*sizeof(double));
C    = (double*) malloc(maxmp*nq*sizeof(double));
C0   = (double*) malloc(maxmp*nq*sizeof(double));
tau  = (double*) malloc(ltau*sizeof(double));
work = (double*) malloc(lwork*sizeof(double));
```

```
// generate random matrices A, B, and zero matrix C
psmatgen("N", "N", A, 1, 1, descA, m, k, my_row,
    my_col, num_proc_row, num_proc_col,
    iaseed/*+time(NULL)*/, &status );
if (status < 0) {
  Cblacs_exit( 0 );
  return EXIT_FAILURE;
}
psmatgen("N", "N", B, 1, 1, descB, k, n, my_row,
    my_col, num_proc_row, num_proc_col,
    ibseed/*+time(NULL)*/, &status );
if (status < 0) {
  Cblacs_exit( 0 );
  return EXIT_FAILURE;
}
psmatgen("Z", "N", C, 1, 1, descC, m, n, my_row,
    my_col, num_proc_row, num_proc_col, 0, &status
    );
if (status < 0) {
  Cblacs_exit( 0 );
  return EXIT_FAILURE;
}
pdlacpy_ ( "All", &m, &k, A, &ione, &ione, descA,
    A0, &ione, &ione, descA );
if (i_am==0) fprintf ( outfile, "Matrices A, B, C
    and R successfully created.\n");

len_mat_name = 1;

//*****************************************************
//          FIRST STEP : PRODUCT MATRIX-MATRIX
//*****************************************************

wall_time -= Cdwalltime00( );

if (i_am==0) {
  fprintf ( outfile,
     "\n*************************************************************
     );
  fprintf ( outfile, "\nFirst step : product
     matrix-matrix\n" );
  fprintf ( outfile,
     "*************************************************************
```

```
      );
    fprintf ( outfile, "Matrix A:\n" );
  }
  pdlaprnt2( &m, &k, A, &ione, &ione, descA, &izero,
     &izero, "A", outfile, work, len_mat_name );
  if (i_am==0) fprintf ( outfile, "\nMatrix G:\n" );
  pdlaprnt2( &k, &n, B, &ione, &ione, descB, &izero,
     &izero, "G", outfile, work, len_mat_name );

  pdgemm_ ( "N", "N", &m, &n, &k, &done, A0, &ione,
     &ione, descA, B, &ione, &ione, descB, &done, C,
     &ione, &ione, descC );
  if (i_am==0) fprintf ( outfile, "\nH := A*G\n" );
  pdlaprnt2( &m, &n, C, &ione, &ione, descC, &izero,
     &izero, "H", outfile, work, len_mat_name );

  //*****************************************************
  //            SECOND STEP : QR DECOMPOSITION
  //*****************************************************

  if (i_am==0) {
    fprintf ( outfile,
       "\n*************************************************************
       );
    fprintf ( outfile, "\nSecond step : QR
       decomposition\n" );
    fprintf ( outfile,
       "*************************************************************
       );
    fprintf ( outfile, "Matrix H:\n" );
  }
  pdlaprnt2( &m, &n, C, &ione, &ione, descC, &izero,
     &izero, "H", outfile, work, len_mat_name );

  /* QR decomposition */
  pdgeqrf_( &m, &n, C, &ione, &ione, descC, tau,
     work, &lwork, &info );
  /* Copy in buffer */
  pdlacpy_ ( "All", &m, &n, C, &ione, &ione, descC,
     C0, &ione, &ione, descC );
  /* To print Q we need to explicit it */
  pdorgqr_( &m, &n, &ltau, C, &ione, &ione, descC,
     tau, work, &lwork, &info );
  if(i_am==0) fprintf ( outfile, "\nMatrix B:\n" );
```

```
    pdlaprnt2( &m, &n, C, &ione, &ione, descC, &izero,
       &izero, "B", outfile, work, len_mat_name );


    //*****************************************************
    //            THIRD STEP : C=Q^t*A
    //*****************************************************

    if (i_am==0) {
      fprintf ( outfile,
         "\n*************************************************************
         );
      fprintf ( outfile, "\nThird step : C=Q^t*A\n" );
      fprintf ( outfile,
         "*************************************************************
         );
      fprintf ( outfile, "Matrix C:\n" );
    }
    /* C = Q^t*A */
    pdormqr_( "L", "T", &m, &n, &ltau, C0, &ione,
       &ione, descC, tau, A0, &ione, &ione, descA,
       work, &lwork, &info );
    pdlaprnt2( &n, &k, A0, &ione, &ione, descA,
       &izero, &izero, "C", outfile, work,
       len_mat_name );


    //*****************************************************
    //            CHECK : A=B*C?
    //*****************************************************

    if (i_am==0) {
      fprintf ( outfile,
         "\n*************************************************************
         );
      fprintf ( outfile, "\nCheck : A~B*C\n" );
      fprintf ( outfile,
         "*************************************************************
         );
      fprintf ( outfile, "A-B*C :\n" );
    }
    /* A = A - B*C */
    pdgemm_ ( "N", "N", &m, &k, &n, &dmone, C, &ione,
       &ione, descC, A0, &ione, &ione, descA, &done,
```

```
      A, &ione, &ione, descA );
    pdlaprnt2( &m, &k, A, &ione, &ione, descA, &izero,
      &izero, "A", outfile, work, len_mat_name );
    resid = pdlange_( "F", &m, &k, A, &ione, &ione,
      descA, work );

    wall_time += Cdwalltime00( );

    if(i_am==0) {
      fprintf ( outfile, "\n||A-B*C||_F = " );
      fprintf ( outfile, "%g\n", resid );
      if(n==k && resid < eps) fprintf ( outfile, "It
        seems to be alright : if l == n the
        approximation is eps-exact !\n" );
      else fprintf ( outfile, "I seems to have a
        problem we should get residual == eps machine
        if l == n... You need to debug :-)\n" );
    }

    //*****************************************************
    //                  FINALIZE ALL
    //*****************************************************

    free(A);
    free(B);
    free(C);
    free(work);

    Cblacs_gridexit( contxt );
  }

  if (i_am==0) {
    fprintf ( outfile,
      "\n*************************************************************
      );
    fprintf ( outfile, "\nWall_time was approximately
      %f seconds.\n", wall_time );
    fprintf ( outfile,
      "*************************************************************
      );
    fprintf ( outfile, "End of prototype.");
    fclose ( outfile );
  }
```

```
  Cblacs_exit( 0 );
  return 0;
}


/* prototype.c ends here */
```

Listing 4 – Générateur de matrice

```
// PSMATGEN : Parallel Real Double precision MATrix
    GENerator.
//  Generate (or regenerate) a distributed matrix A
    (or sub-matrix of A).
// Original fortran code from
    http://acts.nersc.gov/scalapack/hands-on/
// Conversion into C by Kelly McQuighan, with
    modifications that don't
// use the psmatgeninc.f file.
// last modified 8/1/13
// - modifs by Olivier Tissot 5/9/2014:
//   * adding zero matrix generator
// Notes from Fortran code
//  =====
//
//  The code is originally developed by David Walker,
    ORNL,
//  and modified by Jaeyoung Choi, ORNL.
//
//  Reference: G. Fox et al.
//  Section 12.3 of "Solving problems on concurrent
    processors Vol. I"
//
//  MATRICES ARE COLUMN-MAJOR!!!
//
//  Usage
//  =====
//
// psmatgen(      double* A,
//                int* DESC,
//                char* AFORM,
//                char* DIAG,
//                int IROFF,
//                int IRNUM,
//                int ICOFF,
```

```
//                int ICNUM,
//                int MYROW,
//                int MYCOL,
//                int NPROW,
//                int NPCOL)
//
//  Arguments
//  =========
//
//  AFORM   (global input) CHARACTER*1
//          if AFORM = "S" : A is returned is a
    symmetric matrix.
//          if AFORM = "H" : A is returned is a
    Hermitian matrix.
//                           "I" : ones along diagonal given
    by DIAG is returned
//          otherwise a random matrix is generated.
//          UNSUPPORTED OPTIONS if AFORM = "T" : A is
    overwritten with the transpose of
//                                                what
    would normally be generated.
//                                     if AFORM = "C" : A is
    overwritten with the conjugate trans-
//                                                pose
    of what would normally be generated.
//
//  DIAG    (global input) CHARACTER*1
//          if DIAG = "D" : A is diagonally dominant.
//          if DIAG = "N" : A is not diagonally
    dominant.
//          if AFORM = "I" then DIAG can also take any
    number between (-m+1) < DIAG < (m-1)
//            note that DIAG must still be enclosed in
    quotes. Also, DIAG < 0 implies
//            ones will be on a subdiagonal, whereas
    DIAG > 0 implies ones will be on
//            a superdiagonal.
//
//  DESC    (global input)
//          Description of the local matrix. Contains
    9 values as follows:
//          DESC[0] = DTYPE
//          DESC[1] = CTXT
//          DESC[2] = M
```

```
//            DESC[3] = N
//            DESC[4] = MB
//            DESC[5] = NB
//            DESC[6] = RSRC
//            DESC[7] = CSRC
//            DESC[8] = LLD
//
//  M         (global input) INTEGER
//            The number of rows in the generated
    distributed matrix.
//
//  N         (global input) INTEGER
//            The number of columns in the generated
    distributed
//            matrix.
//
//  MB        (global input) INTEGER
//            The row blocking factor of the distributed
    matrix A.
//
//  NB        (global input) INTEGER
//            The column blocking factor of the
    distributed matrix A.
//
//  LLD       (local input) INTEGER
//            The leading dimension of the array
    containing the local
//            pieces of the distributed matrix A.
//
//  A         (local output) REAL             , pointer
    into the local
//            memory to an array of dimension ( LDA, * )
    containing the
//            local pieces of the distributed matrix.
//
//  IA    (global input) INTEGER
//        The global initial row index of the submatrix
    to be generated
//        Note: as written, this function only supports
    if IA is a multiple
//        of mb
//
//  JA    (global input) INTEGER
//            The global initial column index of the
```

```
    submatrix to be generated
//          Note: as written, this function only
    supports if JA is a multiple
//          of nb
//
//  IAROW   (global input) INTEGER
//          The row processor coordinate which holds
    the first block
//          of the distributed matrix A.
//
//  IACOL   (global input) INTEGER
//          The column processor coordinate which
    holds the first
//          block of the distributed matrix A.
//
//  IRNUM   (local input) INTEGER
//          The number of local rows to be generated.
//
//  ICNUM   (local input) INTEGER
//          The number of local columns to be
    generated.
//
//  MYROW   (local input) INTEGER
//          The row process coordinate of the calling
    process.
//
//  MYCOL   (local input) INTEGER
//          The column process coordinate of the
    calling process.
//
//  NPROW   (global input) INTEGER
//          The number of process rows in the grid.
//
//  NPCOL   (global input) INTEGER
//          The number of process columns in the grid.
//
//
    ============================================================


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>
```

```c
#include <blacs_headers.h>
#include <pblas_headers.h>
#include <scalapack_tools_headers.h>
#include "psmatgen.h"
#include "stringTools.h"
#include "pblasIOtools.h"
//#include "psmatgeninc.h"
//#include "mkl_scalapack.h"

static int imax( int a, int b ){
  if (a>b) return(a); else return(b);
}

static double dabs( double a) {
  if (a>0) return a; else return (-1*a);
}

void psmatgen( char* aform, char* diag,  double* A,
   int ia, int ja, int* desc, int msub, int nsub, int
   myrow, int mycol, int nprow, int npcol, int iseed,
   int* status){
  int m, n, mb, nb, lld, rsrc, csrc, i_am, moffglob,
     noffglob, irnum, icnum;
  int mp, SYM, HERM, DIAGDOM, NDIAG, NOTRAN, EYE,
     ZERO, info = 0, infoDesc=0;
  int mrrow, mrcol, moff, noff, mend, nend;
  int jk, ic, ioffc, i, ir, ik, ioffr, j, offset=0;
  int mendglob, nendglob;
  double one = 1.0e0, two = 2.0e0, zero=0.0e0, maxmn;
  double dnb, dmb, dia, dja, dmsub, dnsub, dnprow,
     dnpcol;
  double dmoffglob, dnoffglob, dmendglob, dnendglob;
  int extramb, extranb;

  i_am = myrow*nprow+mycol;
  // get necessary variables from desc
  //ctxt = desc[1];
  m = desc[2];
  n = desc[3];
  mb = desc[4];
  nb = desc[5];
  rsrc = desc[6];
  csrc = desc[7];
```

```
lld = desc[8];
ia--; ja--; // because indeces for C start at zero

// test the input arguments
// rsrc gives the process row over which the first
   row of the matrix is distributed
// csrc gives the process column over which the
   first column of the matrix is distributed
mp = numroc_( &m, &mb, &myrow, &rsrc, &nprow );
//nq = numroc_(&n, &nb, &mycol, &csrc, &npcol );

SYM = ( (*aform == 'S') || (*aform == 's') );
HERM = ( (*aform == 'H') || (*aform == 'h') );
EYE = ( (*aform == 'I') || (*aform == 'i') );
ZERO = ( (*aform == 'Z') || (*aform == 'z') );
NOTRAN = ( (*aform == 'N') || (*aform == 'n') );
DIAGDOM = ( (*diag == 'D') || (*diag == 'd') );
NDIAG = ( (*diag == 'N') || (*diag == 'n') );

if ( (mb != nb )&&(SYM || HERM) ) {
  if ( i_am==0 ) fprintf ( stdout, "Symmetric
     matrices with rowNB not equal to colNB is not
     supported! Creating random matrix instead.\n" );
  SYM = 0; HERM = 0; EYE = 0; NOTRAN = 1;
}
if ( (m != n )&& EYE ) {
  if ( i_am==0 ) fprintf ( stdout, "Identity matrix
     with N not equal to M is not supported!
     Creating random matrix instead.\n" );
  SYM = 0; HERM = 0; EYE = 0; NOTRAN = 1;
}
if ( !DIAGDOM && !NDIAG ) offset=atoi(diag);

if (!(SYM || HERM || NOTRAN || EYE || ZERO) ) info=1;
else if (!DIAGDOM && !NDIAG && !EYE ) info=2;
else if ( EYE && (abs(offset) >= m) ) info=2;
else if ( SYM || HERM ) {
  if (m != n) { info=6; infoDesc=4; }
  else if (mb != nb) {info=6; infoDesc=6; }
}
else if (m < 0) { info = 6; infoDesc=3; }
else if (n < 0) { info = 6; infoDesc=4; }
else if (mb < 1) { info = 6; infoDesc=5; }
else if (nb < 1) { info = 6; infoDesc=6; }
```

```
  else if (lld < 0 ) { info = 6; infoDesc=9; }
  else if ( (rsrc < 0) || (rsrc >= nprow) ) { info=6;
     infoDesc=7; }
  else if ( (csrc < 0) || (csrc >= npcol) ) { info=6;
     infoDesc=8; }
  else if ( (ia % mb) > 0) info=4;
  else if ( msub > (m-ia) ) info=7;
  else if ( (ja % mb) > 0) info=5;
  else if ( nsub > (n-ja) ) info=8;
  else if ( (myrow < 0) || (myrow >= nprow) ) info=9;
  else if ( (mycol < 0) || (mycol >= npcol) ) info=10;

  if ( info != 0) {
    if ( info == 6 ) {
      fprintf ( stdout, "Processor {%d, %d}: On entry
         to psmatgen.c, parameter number %d, which is
         an array, had an illegal value.\n", myrow,
         mycol, info ) ;
      fprintf ( stdout, "In particular, the %dth
         element of DESC is invalid.\n", infoDesc );
    } else if ( info == 1 ) {
      if ( (myrow==rsrc) && (mycol==csrc) ){
      fprintf ( stdout, "Processor {%d, %d}: On entry
         to psmatgen.c, parameter number %d had an
         illegal value.\n", myrow, mycol, info ) ;
      fprintf ( stdout, "Currently supported options
         are: ''N'', ''S'', ''H'', ''I'', ''Z''.\n");
      }
    } else if ( info == 2 ) {
      if ( (myrow==rsrc) && (mycol==csrc) ){
      fprintf ( stdout, "Processor {%d, %d}: On entry
         to psmatgen.c, parameter number %d had an
         illegal value.\n", myrow, mycol, info ) ;
      fprintf ( stdout, "Currently supported options
         are: ''D'', ''N''.\n");
      fprintf ( stdout, "If AFORM=''I'' then DIAG may
         take a number between (-m+1) < DIAG <
         (m-1).\n");
      fprintf ( stdout, "Note that the values of DIAG
         must still be passed as a string:
         ''DIAG''.\n");
      }
    } else {
      fprintf ( stdout, "Processor {%d, %d}: On entry
```

```
            to psmatgen.c, parameter number %d had an
            illegal value.\n", myrow, mycol, info ) ;
    }
    *status = -1;
    return;
  }

  dnb = (double) nb; dmb = (double) mb;
  dia = (double) ia; dja = (double) ja;
  dmsub = (double) msub; dnsub = (double) nsub;
  dnprow = (double) nprow; dnpcol = (double) npcol;

  // parameter values are valid so set some local
     constants
  mrrow = (nprow + myrow - rsrc) % nprow; // my
     process row within matrix distribution
  mrcol = (npcol + mycol - csrc) % npcol; // my
     process column within matrix distribution
  moffglob = floor ( dia / dmb );
  noffglob = floor ( dja / dnb );
  mendglob = ceil ( (dia+dmsub) / dmb );
  nendglob = ceil ( (dja+dnsub) / dnb );
// double dmendglob, dnoffglob, dnendglob, dmoffglob;
  dmoffglob = (double) moffglob; dnoffglob = (double)
     noffglob;
  dmendglob = (double) mendglob; dnendglob = (double)
     nendglob;
  //  fprintf(stdout, "I am {%d,%d}. Moffglob=%d,
     noffglob=%d, mendglob=%d, nendglob=%d.\n", myrow,
     mycol, moffglob, noffglob, mendglob, nendglob);

  moff   = floor( dmoffglob / dnprow ); // my block
     number for row offset
  noff   = floor( dnoffglob / dnpcol ); // my block
     number for column offset
  if ( mrrow < ( moffglob % nprow) ) moff++;
  if ( mrcol < ( noffglob % npcol) ) noff++;
  mend   = ceil( dmendglob / dnprow ); // ending row
     block index, local # of blocks
  nend   = ceil( dnendglob / dnpcol ); // end column
     block index, local # of blocks
  if ( mrrow > ( (mendglob+nprow-1) % nprow ) ) mend--;
  if ( mrcol > ( (nendglob+npcol-1) % npcol ) ) nend--;
```

```
irnum = (mend-moff)*mb; icnum = (nend-noff)*nb;
if ( (msub % mb) != 0 ) {
  extramb = ceil(dmsub/dmb);
  if ( ( (extramb+nprow-1) % nprow ) == mrrow )
     irnum+= (msub % mb);
}
if ( (nsub % nb) != 0 ) {
  extranb = ceil(dnsub/dnb);
  if ( ( (extranb+npcol-1) % npcol ) == mrcol )
     icnum+= (nsub % nb);
}

//  fprintf(stdout, "I am {%d, %d}. Irnum=%d,
    icnum=%d. Moff=%d, mend=%d, noff=%d, nend=%d.\n",
    myrow, mycol, irnum, icnum, moff, mend, noff,
    nend);

// symmetric of Hermitian matrix will be generated
if ( SYM || HERM ) {

  // first generate low triangular part

  // first generate lower triangular part


  jk = 0; // keeps track of column
  for ( ic = noff; ic < nend; ic++ ){
    if (jk > icnum) break;
    ioffc = (ic*npcol+mrcol)*nb;
    for (i = 0; i < nb; i++ ) {
      if (jk > icnum) break;
      ik = 0; //keeps track of rows
      for ( ir = moff; ir < mend; ir++ ){
        if (ik > irnum ) break;
        ioffr = (ir*nprow+mrrow) * mb;
        for ( j = 0; j < mb; j++ ) {
          if (ik > irnum ) break;
          if ( (ioffr+j+offset) >= (ioffc+i) ) {
            srand( iseed+ioffr+i+m*(ioffc+j) );
            *(A+(jk+noff*nb)*mp+ik+moff*mb) = one -
               two*rand()/RAND_MAX;
          } else {
            srand( iseed+(ioffr+i)*n+ioffc+j );
            *(A+(jk+noff*nb)*mp+ik+moff*mb) = one -
```

```
                    two*rand()/RAND_MAX;
                }
                ik=ik+1;
            }
          }
          jk = jk+1;
        }
      }

    } // end if symmetric matrix

    // a random matrix is generated
    else if ( EYE ) {

      jk = 0; // keeps track of column
      for ( ic = noff; ic < nend; ic++ ){
        if (jk > icnum) break;
        ioffc = (ic*npcol+mrcol)*nb;
        for (i = 0; i < nb; i++ ) {
          if (jk > icnum) break;
          ik = 0; //keeps track of rows
          for ( ir = moff; ir < mend; ir++ ){
            if (ik > irnum ) break;
            ioffr = (ir*nprow+mrrow) * mb;
            for ( j = 0; j < mb; j++ ) {
              if (ik > irnum ) break;
              if ( (ioffr+j+offset) == (ioffc+i) ) {
                *(A+(jk+noff*nb)*mp+ik+moff*mb) = one;
              } else {
                *(A+(jk+noff*nb)*mp+ik+moff*mb) = zero;
              }
              ik=ik+1;
            }
          }
          jk = jk+1;
        }
      }

    } // end identity matrix
    else if( ZERO ) {

      jk = 0; // keeps track of column
      for ( ic = noff; ic < nend; ic++ ){
        if (jk > icnum) break;
```

```
      ioffc = (ic*npcol+mrcol)*nb;
      for (i = 0; i < nb; i++ ) {
        if (jk > icnum) break;
        ik = 0; //keeps track of rows
        for ( ir = moff; ir < mend; ir++ ){
          if (ik > irnum ) break;
          ioffr = (ir*nprow+mrrow) * mb;
          for ( j = 0; j < mb; j++ ) {
            if (ik > irnum ) break;
            *(A+(jk+noff*nb)*mp+ik+moff*mb) = zero;
            ik=ik+1;
          }
        }
        jk = jk+1;
      }
    }
  } //end else: zero matrix
  else {

    // initialize random number generator
    srand(i_am+iseed);

    jk = 0; // keeps track of column
    for ( ic = noff; ic < nend; ic++ ){
      if (jk > icnum) break;
      ioffc = (ic*npcol+mrcol)*nb;
      for (i = 0; i < nb; i++ ) {
        if (jk > icnum) break;
        ik = 0; //keeps track of rows
        for ( ir = moff; ir < mend; ir++ ){
          if (ik > irnum ) break;
          ioffr = (ir*nprow+mrrow) * mb;
          for ( j = 0; j < mb; j++ ) {
            if (ik > irnum ) break;
            *(A+(jk+noff*nb)*mp+ik+moff*mb) = one -
                two*rand()/RAND_MAX;
            ik=ik+1;
          }
        }
        jk = jk+1;
      }
    }
  } //end else: random matrix
```

```
  // diagonally dominant matrix will be generated
  if ( DIAGDOM ) {

    if (mb != nb ) {
      if (i_am==0) fprintf ( stdout, "Diagonally
         dominant matrices with rowNB not equal to
         colNB is not supported! Command ignored.\n" );
      return;
    }

    if (EYE) {
      if (i_am==0) fprintf ( stdout, "Identity matrix
         is automatically diagonally dominant! Command
         ignored.\n" );
      return;
    }
  // since the magnitude of each element is less than
     1, adding the max( n, m ) to the diagonal is
     sufficient
  // Def: diagonally dominant if the magnitude of the
     diagonal element in a row (column) is greater than
  //       the sum of magnitude of all other elements
     in the row (column). In other words,
  //       ||a_{ii}|| >= \Sum_{j} a_{ij}, \Sum_{j}
     a_{ji}
    maxmn = (double) imax(m,n);
    jk = 0; // keeps track of column
    for ( ic = noff; ic < nend; ic++ ){
      if (jk > icnum) break;
      ioffc = (ic*npcol+mrcol)*nb;
      for (i = 0; i < nb; i++ ) {
        if (jk > icnum) break;
        ik = 0; //keeps track of rows
        for ( ir = moff; ir < mend; ir++ ){
          if (ik > irnum ) break;
          ioffr = (ir*nprow+mrrow) * mb;
          for ( j = 0; j < mb; j++ ) {
            if (ik > irnum ) break;
            if ( (ioffr+j+offset) == (ioffc+i) ) {
//              fprintf(stdout, "I am {%d,%d}. ic=%d,
   ir=%d, ioffr=%d, ioffc=%d.\n", myrow, mycol, ic,
   ir, ioffr, ioffc);
              *(A+(jk+noff*nb)*mp+ik+moff*mb) =
                 dabs(*(A+(jk+noff*nb)*mp+ik+moff*mb))
```

```
                    + maxmn;
                }
                ik=ik+1;
            }
        }
        jk = jk+1;
    }
  }

  return;
}
```

Listing 5 – Lecteur du fichier de paramètres

```
// -- PBLAS Example code --
// Original fortran code from
   http://acts.nersc.gov/scalapack/hands-on/
// Conversion into C by Kelly McQuighan,
// last modified 7/31/13
// - modifs by Olivier Tissot 5/9/2014 :
//   * fix bug : bad syntax for broadcast receive
// reads system parameters from the file BLAS.dat

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>
#include <blacs_headers.h>
#include <pblas_headers.h>
#include "pspblasinfo.h"
#include <string.h>
#include "stringTools.h"
#include "pblasIOtools.h"
#include "nameFile.h"

// returns -1 if unsuccessful, 0 if successful
void pspblasinfo( int* m, int* n, int* k, int*
    block_size, int* num_proc_row, int* num_proc_col,
    int* sys_param, int* i_am, int* num_procs, char
    folder[], const char * infile_name, char*
    outfile_name, char* usr_info, int*
```

```
 outfile_name_len, int* status ){

char line[100], info[100], trash[100];
char suffix_out[]="out", file_prefix[100];
int contxt, i;
FILE *infile;

Cblacs_get( -1, 0, &contxt );
Cblacs_gridinit( &contxt, "Row-major", 1, *num_procs
   );

// processor 0 reads data from PBLAS.dat and
   broadcasts to other processors.
if ( *i_am==0 ) {
  infile = fopen(infile_name, "r");
  if (infile == NULL) {
    fprintf(stdout,
       "*******************************************************\n")
    fprintf(stdout, "Error: input file %s does not
       exist !\n", infile_name);
    fprintf(stdout,
       "*******************************************************\n")
    *status = -1;
    Cigebs2d( contxt, "All", " ", 1, 1, status, 1 );
    return;
  }
  // header line
  if (fgets(line, 100, infile)==NULL ) {
    fprintf(stdout,
       "\n*******************************************************\n"
    fprintf(stdout, "Error: input file does not
       contain any data!\n");
    fprintf(stdout,
       "\n*******************************************************\n"
    *status = -1;
    Cigebs2d( contxt, "All", " ", 1, 1, status, 1 );
    return;
  }
  // user info
  if (fgets(usr_info, 100, infile)==NULL ) {
    fprintf(stdout,
       "\n*******************************************************\n"
    fprintf(stdout, "Error: input file does not
       contain any data!\n");
```

```
      fprintf(stdout,
         "\n*****************************************************\n
      *status = -1;
      //Cigebs2d( contxt, "All", " ", 1, 1, status, 1
         );
      return;
    }
    // output file name
    if (fgets(line, 100, infile)==NULL )  {
      fprintf(stdout,
         "\n*****************************************************\n
      fprintf(stdout, "Error: input file should
         contain the output file name on the 3rd
         line!\n");
      fprintf(stdout,
         "\n*****************************************************\n
      *status = -1;
      Cigebs2d( contxt, "All", " ", 1, 1, status, 1 );
      return;
    }
    if ( sscanf(line,"%[^\t]%[^\n]", file_prefix,
     trash) < 1 ) {
      fprintf(stdout,
         "\n*****************************************************\n
      fprintf(stdout, "Error: input file not formatted
         correctly! 3rd line should read (output file
         name) (tab) (additional text).\n");
      fprintf(stdout,
         "\n*****************************************************\n
      *status = -1;
      Cigebs2d( contxt, "All", " ", 1, 1, status, 1 );
      return;
    }

    //  system parameters
    for (i=0; i<6; i++ ) {

      if (fgets(line, 100, infile) == NULL ) {
        fprintf(stdout,
           "\n*****************************************************
        fprintf(stdout, "Error: input file not
           formatted correctly! Lines 4-13 should
           contain Scalapack matrix information in the
           format\n\n(value of m) (tab) (additional
```

```
            text)\n");
        fprintf(stdout, "in the following order: m, n,
            nrhs, mb, nb, nb_rhs, max_lldA, max_lldB,
            nprow, npcol\n");
        fprintf(stdout,
            "\n*****************************************************\
        *status = -1;
        Cigebs2d( contxt, "All", " ", 1, 1, status, 1
            );
      return;
      }
      if ( sscanf(line,"%[^\t]%[^\n]", info, trash) <
         2 ) {
        fprintf(stdout,
            "\n*****************************************************\
        fprintf(stdout, "Error: input file not
            formatted correctly on %dth line! It should
            read\n\n(value) (tab) (additional text)\n",
            i+4);
        fprintf(stdout,
            "\n*****************************************************\
        *status = -1;
        Cigebs2d( contxt, "All", " ", 1, 1, status, 1
            );
        return;
      }
      sys_param[i] = atoi(info);
    }

    // completed successfully. broadcast parameters to
        other processes
    *status = 0;
    Cigebs2d( contxt, "All", " ", 1, 1, status, 1 );
    Cigebs2d( contxt, "All", " ", 6, 1, sys_param, 6);

    fclose(infile);

  } else {

    // receive system variables and store in
        corresponding variables
    Cigebr2d( contxt, "All", " ", 1, 1, status, 1, 0,
       0 );
    if ( *status < 0 ) return;
```

```
    Cigebr2d( contxt, "All", " ", 6, 1, sys_param, 6,
        0, 0 );

  } // end if-then-else

  // store system parameters in their appropriate
      values
  *m             = sys_param[0];
  *n             = sys_param[1];
  *k             = sys_param[2];
  *block_size    = sys_param[3];
  *num_proc_row  = sys_param[4];
  *num_proc_col  = sys_param[5];

  if (*i_am == 0) {
    nameFile ( folder, file_prefix, "", suffix_out,
        outfile_name );
  }
  Cblacs_gridexit( contxt );

  return;
}
```