REPUBLIQUE DU BENIN

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE (MESRS)

UNIVERSITÉ NATIONALE DES SCIENCES, TECHNOLOGIES, INGÉNIERIE ET MATHÉMATIQUES (UNSTIM)

ÉCOLE NATIONALE SUPÉRIEURE DES GÉNIE MATHÉMATIQUE ET MODÉLISATION (ENSGMM-ABOMEY)

UTILSATION DE INTEL MKL EN CALCUL SCIENTIFIQUE : BLAS ET LAPACK

Membres du groupe :

AHOTONHOUN Aimé Césaire

BONOU Justus

HANDJEMEDJI Ezéchiel

PROJET DE CALCUL SCIENTIFIQUE

1^{re} année - CYCLE D'INGÉNIEUR

Sous la supervision de : Dr. Ing. Carlos AGOSSOU

ANNÉE ACADÉMIQUE : 2025-2026

Table des matières

1 Introduction à Intel Math Kernel Library (MKL)

1.1 Présentation générale

Intel Math Kernel Library (MKL) est une bibliothèque mathématique optimisée pour les processeurs Intel. Elle fournit des implémentations haute performance des routines BLAS (Basic Linear Algebra Subprograms), LAPACK (Linear Algebra PACKage), ScaLAPACK, ainsi que des solveurs pour les équations différentielles, les transformées de Fourier et les problèmes d'optimisation. MKL est particulièrement adaptée aux calculs scientifiques intensifs grâce à son optimisation pour l'architecture x86-64 et son support du parallélisme multithread via OpenMP et TBB.

1.2 Applications pratiques de BLAS et LAPACK dans la vie active

Les bibliothèques BLAS et LAPACK, via Intel MKL, sont omniprésentes dans de nombreux domaines professionnels et industriels :

1.2.1 Ingénierie et Simulation Numérique

Dans le domaine de l'ingénierie aéronautique, par exemple, les simulations de dynamique des fluides computationnelle (CFD) reposent massivement sur la résolution de systèmes linéaires de grande taille. Les équations de Navier-Stokes, discrétisées sur des maillages complexes de plusieurs millions de points, génèrent des matrices creuses dont la manipulation efficace nécessite des routines BLAS optimisées. La stabilité et la précision numérique offertes par LAPACK sont cruciales pour garantir la fiabilité des prédictions aérodynamiques.

1.2.2 Finance Quantitative

Le secteur financier utilise intensivement ces bibliothèques pour la modélisation des risques, l'optimisation de portefeuilles, et le pricing d'options exotiques. Les méthodes de Monte-Carlo, les algorithmes de réduction de dimensionnalité, et les calculs de corrélations entre actifs impliquent des opérations matricielles massives. La performance d'Intel MKL permet de réduire significativement les temps de calcul, un avantage compétitif décisif dans les marchés financiers où chaque milliseconde compte.

1.2.3 Recherche Scientifique

En physique des matériaux, la résolution de l'équation de Schrödinger pour des systèmes complexes nécessite le calcul de valeurs propres de matrices de très grande dimension. En imagerie médicale, la reconstruction tomographique repose sur des algorithmes de décomposition en valeurs singulières (SVD) fournis par LAPACK. Dans tous ces domaines, la robustesse numérique et la performance d'Intel MKL sont essentielles.

1.2.4 Intelligence Artificielle et Apprentissage Automatique

Les réseaux de neurones profonds impliquent des millions d'opérations matricielles lors de l'entraînement et de l'inférence. Bien que des bibliothèques spécialisées comme cuDNN existent pour les GPU, de nombreuses étapes de prétraitement et de post-traitement sur CPU utilisent intensivement BLAS et LAPACK. L'analyse en composantes principales (PCA), méthode fondamentale de réduction de dimensionnalité, repose directement sur le calcul de vecteurs propres via LAPACK.

1.3 Installation et configuration

1.3.1 Installation sur Linux

```
# Méthode 1 : Via le gestionnaire de paquets
sudo apt-get install intel-mkl

# Méthode 2 : Téléchargement depuis le site Intel
# 1. Télécharger le package depuis intel.com
# 2. Extraire et exécuter l'installateur
tar -xzvf l_mkl_2024.0.0.tgz
cd l_mkl_2024.0.0
sudo ./install.sh

# Configuration des variables d'environnement
export MKLROOT=/opt/intel/mkl
export LD_LIBRARY_PATH=$MKLROOT/lib/intel64:$LD_LIBRARY_PATH
```

1.3.2 Installation sur Windows

— Télécharger Intel oneAPI Base Toolkit depuis le site Intel
— Exécuter l'installateur graphique
— Configurer les variables d'environnement dans Visual Studio

1.3.3 Liaison avec un programme C/C++

```
# Compilation avec gcc
gcc -o mon_programme mon_programme.c -lmkl_rt -lpthread -lm -ldl

# Avec Intel Compiler
icc -o mon_programme mon_programme.c -mkl
```

2 Problème physique : Vibrations d'une membrane non uniforme

2.1 Contexte physique

On s'intéresse à une membrane carrée de dimension 1×1 mètre, tendue et fixée sur tout son pourtour. Cette membrane peut représenter divers systèmes physiques :
— Un **drum** ou instrument de percussion
— Un **capteur vibratoire** en ingénierie
— Une **plaque instrumentale** en acoustique
La particularité de cette membrane réside dans sa **non-uniformité** :
— La tension varie spatialement sur le domaine
— La densité de masse n'est pas constante
— Un obstacle central modifie localement la rigidité

2.2 Modèle mathématique continu

Le comportement vibratoire est décrit par l'équation aux dérivées partielles suivante :

$$-\nabla \cdot \big(p(x,y)\nabla u(x,y)\big) + q(x,y)u(x,y) = \lambda\, w(x,y)\, u(x,y), \quad (x,y) \in \Omega$$

avec :
— $\Omega = [0,1] \times [0,1]$: domaine carré
— $u(x,y)$: déplacement vertical (mode propre)
— $\lambda = \omega^2$: valeur propre (carré de la fréquence angulaire)
— $p(x,y)$: coefficient de tension variable
— $w(x,y)$: densité de masse variable
— $q(x,y)$: potentiel représentant l'obstacle central

2.3 Conditions aux limites et données

2.3.1 Conditions aux limites

Dirichlet homogène sur tout le bord :

$$u(x,y) = 0 \quad \text{sur } \partial\Omega$$

2.3.2 Fonctions coefficients

$$\begin{aligned} p(x,y) &= 1 + 0.5\sin(2\pi x)\cos(2\pi y) \\ w(x,y) &= 1 + 0.3xy \\ q(x,y) &= 50e^{-50[(x-0.5)^2+(y-0.5)^2]} \end{aligned}$$

3 Discrétisation par différences finies

3.1 Maillage du domaine

On considère un maillage uniforme de $N \times N$ points intérieurs. Le pas spatial est :

$$h = \frac{1}{N+1}$$

Les coordonnées des points sont :

$$x_i = i \cdot h, \quad y_j = j \cdot h, \quad i,j = 0,1,\ldots,N+1$$

Les points intérieurs (inconnues) sont pour $i,j = 1,\ldots,N$.

3.2 Discrétisation des opérateurs différentiels

3.2.1 Opérateur Laplacien avec coefficient variable

L'opérateur $\nabla \cdot (p\nabla u)$ est discrétisé en utilisant un schéma aux différences finies centrées :

$$\nabla\cdot(p\nabla u) \approx \frac{1}{h^2}\left[p_{i+\frac{1}{2},j}(u_{i+1,j}-u_{i,j}) - p_{i-\frac{1}{2},j}(u_{i,j}-u_{i-1,j})\right] + \frac{1}{h^2}\left[p_{i,j+\frac{1}{2}}(u_{i,j+1}-u_{i,j}) - p_{i,j-\frac{1}{2}}(u_{i,j}-u_{i,j-1})\right]$$

où $p_{i+\frac{1}{2},j} = \frac{p_{i,j}+p_{i+1,j}}{2}$ et $p_{i,j+\frac{1}{2}} = \frac{p_{i,j}+p_{i,j+1}}{2}$.

3.2.2 Discrétisation complète

Pour chaque point intérieur (i, j), l'équation discrétisée s'écrit :

$$-\frac{1}{h^2}\left[p_{i+\frac{1}{2},j}(u_{i+1,j}-u_{i,j})-p_{i-\frac{1}{2},j}(u_{i,j}-u_{i-1,j})\right] - \frac{1}{h^2}\left[p_{i,j+\frac{1}{2}}(u_{i,j+1}-u_{i,j})-p_{i,j-\frac{1}{2}}(u_{i,j}-u_{i,j-1})\right] + q_{i,j}u_{i,j} = \lambda w_{i,j}u_{i,j} \quad (1)$$

3.3 Formulation matricielle

3.3.1 Numérotation des inconnues

On utilise une numérotation lexicographique (ligne par ligne) :

$$k = (i-1)\times N + (j-1), \quad k = 0, \ldots, N^2-1$$

3.3.2 Construction des matrices

On obtient un problème aux valeurs propres généralisé de la forme :

$$A\mathbf{u} = \lambda B\mathbf{u}$$

où :
— $\mathbf{u} \in \mathbb{R}^{N^2}$: vecteur des déplacements aux nœuds
— $A \in \mathbb{R}^{N^2\times N^2}$: matrice de rigidité (creuse, symétrique)
— $B \in \mathbb{R}^{N^2\times N^2}$: matrice de masse (diagonale)

3.3.3 Structure de la matrice A

Pour un nœud intérieur (i, j), les contributions non nulles sont :

$$\begin{aligned}
A_{k,k} &= \frac{1}{h^2}(p_{i+\frac{1}{2},j}+p_{i-\frac{1}{2},j}+p_{i,j+\frac{1}{2}}+p_{i,j-\frac{1}{2}})+q_{i,j}\\
A_{k,k+1} &= -\frac{1}{h^2}p_{i,j+\frac{1}{2}} \quad (\text{si } j<N)\\
A_{k,k-1} &= -\frac{1}{h^2}p_{i,j-\frac{1}{2}} \quad (\text{si } j>1)\\
A_{k,k+N} &= -\frac{1}{h^2}p_{i+\frac{1}{2},j} \quad (\text{si } i<N)\\
A_{k,k-N} &= -\frac{1}{h^2}p_{i-\frac{1}{2},j} \quad (\text{si } i>1)
\end{aligned}$$

3.3.4 Matrice de masse B

La matrice B est diagonale avec :

$$B_{k,k} = w_{i,j}$$

4 Solveurs Intel MKL pour problèmes aux valeurs propres

4.1 Classification des solveurs disponibles

Intel MKL propose plusieurs familles de solveurs pour les problèmes aux valeurs propres :

Solveur	Type de matrice	Problème	Fonction MKL
`dsyev`	Dense, symétrique	Standard	`LAPACK_dsyev`
`dsyevd`	Dense, symétrique	Standard (diviser-pour-regner)	`LAPACK_dsyevd`
`dsygv`	Dense, symétrique	Généralisé	`LAPACK_dsygv`
`dsygvd`	Dense, symétrique	Généralisé (diviser-pour-regner)	`LAPACK_dsygvd`
FEAST	Creuse, symétrique	Standard/Généralisé	`mkl_sparse_ee`
PARDISO+	Creuse, symétrique	Standard	`pardiso_eigensolver`

TABLE 1 – Solveurs de valeurs propres dans Intel MKL

4.2 Solveur `dsygv` pour matrices denses

4.2.1 Description

Le solveur `dsygv` résout le problème aux valeurs propres généralisé $A\mathbf{x} = \lambda B\mathbf{x}$ pour des matrices symétriques réelles. Il utilise la réduction au problème standard suivante de l'algorithme QR.

4.2.2 Interface en C

```
void dsygv(
    int *itype,      // Type de problème (1: A*x = lambda*B*x)
    char *jobz,      // 'N': valeurs propres seulement, 'V': vecteurs aussi
    char *uplo,      // 'U': partie supérieure stockée, 'L': partie inférieure
    int *n,          // Ordre des matrices
    double *A,       // Matrice A
    int *lda,        // Leading dimension de A
    double *B,       // Matrice B
    int *ldb,        // Leading dimension de B
    double *w,       // Valeurs propres
    double *work,    // Tableau de travail
    int *lwork,      // Taille de work
    int *info        // Information de sortie
);
```

4.2.3 Complexité algorithmique

— Réduction à la forme standard : $O(n^3)$
— Algorithme QR : $O(n^3)$
— Stockage : $O(n^2)$

4.3 Solveur FEAST pour matrices creuses

4.3.1 Principe de la méthode FEAST

FEAST (Fast Eigensolver using Spectral Transformation) est un solveur basé sur la théorie des résidus pour extraire les valeurs propres dans un intervalle donné. Il combine :

1. Une transformation spectrale
2. Une intégration de contour
3. Une projection sur un sous-espace de Rayleigh

4.3.2 Avantages pour notre problème

— Extraction sélective des valeurs propres (premières k valeurs)
— Efficacité mémoire pour les matrices creuses
— Parallélisation naturelle

4.3.3 Interface MKL

```
sparse_status_t mkl_sparse_ee(
    sparse_matrix_t A,           // Matrice creuse A
    sparse_matrix_t B,           // Matrice creuse B (optionnel)
    sparse_operation_t operation,// Opération sur A
    struct matrix_descr descr,   // Descripteur de matrice
    MKL_INT k0,                  // Nombre de valeurs à calculer
    MKL_INT *k,                  // Nombre de valeurs trouvées
    double *E,                   // Valeurs propres
    double *X,                   // Vecteurs propres
    double *res                  // Résidus
);
```

5 Application au problème de la membrane

5.1 Choix du solveur adapté

Pour notre problème de membrane avec $N = 100$ (soit 10^4 degrés de liberté), les considérations suivantes guident notre choix :

Solveur	Stockage	Complexité	Suitabilité
`dsygv`	8×10^8 octets	$O(10^{12})$	Non adapté
FEAST	$\sim 5 \times 10^6$ octets	$O(10^7)$	Très adapté

TABLE 2 – Comparaison des solveurs pour $N = 100$

Le solveur FEAST est donc choisi pour sa capacité à traiter efficacement les matrices creuses.

5.2 Implémentation conceptuelle

5.2.1 Pseudo-code de résolution

5.3 Analyse de convergence

La précision de la solution discrétisée dépend du pas spatial h. L'erreur d'approximation suit :

$$|\lambda_{exact} - \lambda_h| \leq Ch^2$$

où C est une constante dépendant des dérivées secondes du mode propre.

Algorithm 1 Résolution du problème de membrane avec Intel MKL

Require: N : nombre de points par dimension
Ensure: λ_i : valeurs propres, U_i : modes propres
Définir $h \leftarrow 1/(N+1)$
Allouer mémoire pour matrices A et B au format CSR
Construction de la matrice A
for $i = 1$ **to** N **do**
  for $j = 1$ **to** N **do**
    Calculer $p_{i,j}$, $w_{i,j}$, $q_{i,j}$
    Déterminer les indices des voisins
    Calculer les coefficients $A_{k,k}$, $A_{k,k\pm1}$, $A_{k,k\pm N}$
    Remplir la structure CSR de A
  end for
end for
Construction de la matrice B (diagonale)
Initialiser le solveur FEAST
Définir l'intervalle de recherche $[0, \lambda_{max}]$
Résoudre $A\mathbf{u} = \lambda B\mathbf{u}$ avec FEAST
Extraire les k premières valeurs/vecteurs propres
Post-traitement et visualisation

6 Exemples pratiques d'utilisation de BLAS et LAPACK

6.1 Exercice 1 : Optimisation d'Algorithmes avec BLAS

6.1.1 Énoncé

Vous développez un système de recommandation nécessitant le calcul de similarités entre des vecteurs de caractéristiques. La similarité cosinus entre deux vecteurs A et B est définie par :

$$\text{similarité} = \frac{A \cdot B}{\|A\|\|B\|}$$

Une implémentation naïve montre des performances insuffisantes pour des volumes de données importants.

6.1.2 Tâches :

1. Implémenter une version optimisée avec BLAS niveau 1
2. Comparer les performances avec la version naïve
3. Vérifier la stabilité numérique

6.1.3 Corrigé

La version optimisée repose sur les routines BLAS niveau 1 :
— `cblas_ddot` pour le produit scalaire
— `cblas_dnrm2` pour le calcul des normes
L'utilisation de BLAS permet de déléguer les calculs à des routines vectorisées et optimisées, réduisant significativement le temps d'exécution pour des vecteurs de grande taille.

6.2 Exercice 2 : Solveur de Systèmes Linéaires avec LAPACK

6.2.1 Énoncé

Vous développez un simulateur de circuits électriques nécessitant la résolution de systèmes linéaires issus de la discrétisation des lois de Kirchhoff. Ces systèmes peuvent être de grande taille et mal conditionnés.

6.2.2 Tâches :

1. Résoudre le système avec LAPACK
2. Choisir automatiquement la méthode adaptée
3. Évaluer la qualité numérique de la solution

6.2.3 Corrigé

La résolution repose sur :
— `dgesv` pour les matrices générales (LU)
— `dpotrf/dpotrs` pour les matrices symétriques définies positives (Cholesky)
Un test préalable permet de détecter si la matrice est SDP afin d'utiliser la méthode la plus efficace. La qualité de la solution est évaluée par le calcul du résidu $\|Ax - b\|$.

7 Conclusions

Cette étude a démontré l'efficacité d'Intel MKL pour résoudre les problèmes aux valeurs propres issus de la discrétisation d'EDP. Le solveur FEAST s'est avéré particulièrement adapté aux matrices creuses de grande taille, offrant un excellent compromis vitesse/précision/mémoire.

7.1 Perspectives

— Extension aux problèmes 3D avec maillages non structurés
— Utilisation de la version GPU d'Intel MKL (oneMKL)
— Intégration avec des solveurs non-linéaires pour les grandes déformations
— Application à l'optimisation topologique de membranes

7.2 Recommandations

Pour les utilisateurs d'Intel MKL :

1. Privilégier les formats de stockage creux (CSR) pour les EDP
2. Utiliser FEAST pour l'extraction sélective de valeurs propres
3. Profiler régulièrement les performances avec Intel VTune
4. Tirer parti du parallélisme via OpenMP directives

Références

[1] Intel Corporation, *Intel oneAPI Math Kernel Library Developer Reference*, 2024.

[2] Saad, Y., *Numerical Methods for Large Eigenvalue Problems*, SIAM, 2011.

[3] Trefethen, L. N., *Spectral Methods in MATLAB*, SIAM, 2000.

[4] Polizzi, E., *Density-Matrix-Based Algorithms for Solving Eigenvalue Problems*, Phys. Rev. B, 2009.

[5] Bathe, K. J., *Finite Element Procedures*, Prentice Hall, 2014.

[6] Golub, G. H., & Van Loan, C. F., *Matrix Computations (4th ed.)*, Johns Hopkins University Press, 2013.

[7] Trefethen, L. N., & Bau, D., *Numerical Linear Algebra*, SIAM, 1997.

[8] Anderson, E., et al., *LAPACK Users' Guide (3rd ed.)*, SIAM, 1999.

[9] Dongarra, J. J., et al., *A set of level 3 basic linear algebra subprograms*, ACM Transactions on Mathematical Software, 1990.

[10] Intel Math Kernel Library Developer Reference, `https://software.intel.com/content/www/us/en/develop/documentation/mkl-developer-reference-c/top.html`

[11] Netlib BLAS Documentation, `http://www.netlib.org/blas/`

[12] Netlib LAPACK Documentation, `http://www.netlib.org/lapack/`

[13] LAPACK Working Notes, `http://www.netlib.org/lapack/lawns/`

A Structure CSR (Compressed Sparse Row)

Pour une matrice de taille $n \times n$ avec nnz éléments non nuls :
— `values[nnz]` : valeurs non nulles
— `columns[nnz]` : indices de colonne
— `row_index[n+1]` : indices de début de ligne

B Paramètres FEAST recommandés

— Tolérance : 10^{-10}
— Sous-espace initial : $2\times$ nombre de valeurs cherchées
— Points de quadrature de contour : 8
— Solveur linéaire : PARDISO