



**Rapport du T.P. :  
5A. Mathématiques appliquées et Modélisation 2017-2018 (option C.O. et B.D.P.)**

**Calcul haute performance :  
Méthodes numériques pour le C.H.P. : PetSc**



**Groupe de travail : ABOUADALLAH Mohamed Anwar,**

**École Polytechnique Lyon  
15 Boulevard Latarjet 69622 Villeurbanne**

## Table des matières

<b>1</b>	<b>Introduction :</b>	<b>2</b>
1.1	Parallélisme :	2
1.1.1	L'idée du calcul parallèle :	2
1.2	Objective du T.P. :	2
1.2.1	Équation de la chaleur :	2
1.2.2	Petsc :	3
<b>2</b>	<b>Partie théorique :</b>	<b>3</b>
2.1	Discretisations :	3
2.2	Discretisation de nos problèmes :	4
<b>3</b>	<b>Implementation PETSc :</b>	<b>6</b>
3.1	Présentation :	6
3.2	Présentation du travail qu'on va effectuer :	6
3.2.1	Intégration du problème 1 :	6
3.2.2	Intégration du problème 2 :	6
3.3	Intégration du problème 1 :	6
<b>4</b>	<b>Conclusion :</b>	<b>9</b>

# 1 Introduction :

## 1.1 Parallélisme :

### 1.1.1 L'idée du calcul parallèle

Il existe deux types de programmations : Séquentiel et Parallèle. Dans le modèle séquentiel, un programme est exécuté par un unique processus, malheureusement, divers champs d'application tels la mécanique des fluides ou l'intelligence artificiel imposent une limitation de la puissance de calcul d'un processeur et de la capacité mémoire. D'où l'idée du calcul ou programmation parallèle qui consiste à associer plusieurs processeurs pour traiter un même problème.

En effet, le parallélisme présente plusieurs avantages par rapport au calcul séquentiel parmi eux on peut citer :

- ★ La réduction du temps de réponse Augmentation de la taille des problèmes traités Utilisation des ressources existantes.
- ★ La complexification des modèles de simulation.
- ★ Développement structuré chaque modèle simulé ne connaît que les champs de données et les solveurs qui lui sont nécessaires.
- ★ Une meilleure gestion de la mémoire : effort sur la localisation des données.

Durant cours de calcul haute performance, nous allons étudier trois logiciels très utilisés dans le calcul parallèle : MPI (Message Passing Interface), le openMP (Multithreading) et PetSci. Ainsi, durant ce second T.P., nous allons utiliser PetSc et expliquer le fonctionnement du transfert des messages.

## 1.2 Objective du T.P. :

L'objectif de ce T.P. est de mettre en œuvre une bibliothèque Petsc pour résoudre le problème 3d de l'équation de la chaleur avec des conditions aux limites périodiques en espace par une méthode de décomposition du domaine dans le temps.

### 1.2.1 Équation de la chaleur :

En mathématiques et en physique théorique, l'équation de la chaleur est une équation aux dérivées partielles parabolique, pour décrire le phénomène physique de conduction thermique, introduite initialement en 1807 par Joseph Fourier. Ce type d'équations est utilisé dans divers domaines tels que la physique, la finance (Black-Scholes) ...

En trois dimensions, l'équation de la chaleur permet de voir que l'interaction entre trois zones de températures initiales différentes.

Ainsi, voici le problème 3d de l'équation de la chaleur avec des conditions aux limites périodiques en espace qu'on souhaite résoudre :

Soit  $\Omega = [0, 1]^3$  et  $T > 0$ , alors :

**Problème 1 :**

$$\left\{ \begin{array}{l} \frac{\partial u}{\partial t^2}(t, x, y, z) = \Delta^2(u)(t, x, y, z) - g(t, u(t, x, y, z)), \forall t \in [0, T], \forall (x, y, z) \in \Omega \\ u(0, x, y, z) = u_0(x, y, z), \forall (x, y, z) \in \Omega \\ u(t, x, y, z) = u(t, x + 1, y, z), \forall (x, y, z) \in \Omega \\ u(t, x, y, z) = u(t, x - 1, y, z), \forall (x, y, z) \in \Omega \\ u(t, x, y, z) = u(t, x, y + 1, z), \forall (x, y, z) \in \Omega \\ u(t, x, y, z) = u(t, x, y - 1, z), \forall (x, y, z) \in \Omega \end{array} \right.$$

On transforme ce problème aux valeurs initiales en  $t=0$  en un problème aux conditions aux limites en  $t=0$  et en  $t=T$  en prenant la dérivée partielle en temps. Ainsi notre problème devient : **Problème 2 :**

$$\left\{ \begin{array}{l} \frac{\partial^2 u}{\partial t^2}(t, x, y, z) = \Delta^2(u)(t, x, y, z) + \Delta g(t, u(t, x, y, z)) + \frac{g(t, u(t, x, y, z))}{\partial t}, \forall t \in [0, T], \forall (x, y, z) \in \Omega \\ \frac{\partial u}{\partial t}(t, x, y, z) - \Delta(u)(t, x, y, z) - g(t, u(t, x, y, z)) = 0, \forall t \in [0, T], \forall (x, y, z) \in \Omega \\ u(0, x) = u_0^h(x), \forall (x, y, z) \in \Omega \\ u(t, x, y, z) = u(t, x + 1, y, z), \forall (x, y, z) \in \Omega \\ u(t, x, y, z) = u(t, x - 1, y, z), \forall (x, y, z) \in \Omega \\ u(t, x, y, z) = u(t, x, y + 1, z), \forall (x, y, z) \in \Omega \\ u(t, x, y, z) = u(t, x, y - 1, z), \forall (x, y, z) \in \Omega \end{array} \right.$$

Maintenant qu'on explicité notre équation, on va effectuée notre discrétisation par différences finies d'ordre 2.

### 1.2.2 Petsc :

PETSc est l'acronyme de "Portable, Extensible Toolkit for Scientific Computation" est une bibliothèque de calcul scientifique parallèle principalement écrites en C, elle permet de gérer des vecteurs et des matrices creuses et de résoudre les systèmes linéaires correspondants avec des solveurs directs ou itératifs, elle nous permet aussi de mobiliser des solveurs non-linéaires des méthodes de résolution d'équations différentielles.

Cette bibliothèque est basée sur M.P.I. et blas et permet de mobiliser des librairies externes. Dans la suite de cette section nous allons exposer les principales fonctionnalités de PETSc :

- ★ Elle fournit plusieurs routines pour la résolution des EDP :
  - ✱ Préconditionneurs parallèles évolutifs, y compris les solveurs directs multigrilles et éparpillés
  - ✱ Méthodes de sous-espace Krylov
  - ✱ Solveurs non linéaires parallèles, tels que la méthode de Newton et le GMRES non linéaire
  - ✱ Solveurs pas à pas parallèles
  - ✱ Profilage automatique de l'utilisation de la virgule flottante et de la mémoire.
- ★ Fonctionne sur toute architecture parallèle possédant une implémentation MPI.
- ★ Utilisable en C, C++, Fortran et Python
- ★ Les communications sont cachées à l'utilisateur

Dans la section implémentation nous allons expliquer comment on va utiliser cette librairie.

## 2 Partie théorique :

### 2.1 Discrétisations :

On définit les discrétisation régulière en espace pour x, y, z, pour cela, on commence par définir les pas d'espaces selon les trois directions :  $h_x = 1./N_x$ ,  $h_y = 1./N_y$  et  $h_z = 1./N_z$  ce qui nous permet de définir le maillage de  $N_x \times N_y \times N_z$  de points de coordonnées  $x_i = ih_x, y_j = jh_y, z_k = kh_z$  puis nous allons chercher à approximer  $u_{i,j,k} = u(x_i, y_j, z_k)$

Pour cela, on commence par approcher les opérateurs discret par le théorème de Taylor :  $\frac{\partial^2 u(q)}{\partial q^2} = \frac{u(q+h) - 2u(q) + u(q-h)}{h^2} + O(h^2)$

Ensuite,

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j,k} = \frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{h_x^2} + O(h^2)$$

et

$$\frac{(\partial^4 u)_{i,j,k}}{\partial x^4} = \frac{\partial^2}{\partial x^2} \left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j,k}$$

En injectant notre première formule dans la seconde on obtient :

$$\frac{(\partial^4 u)_{i,j,k}}{\partial x^4} = \frac{\partial^2}{\partial x^2} \left( \frac{u(q+h) - 2u(q) + u(q-h)}{h^2} + O(h^2) \right)$$

Puis on réinjecte notre discrétisation dans notre formule, ce qui nous donne :

$$\frac{(\partial^4 u)_{i,j,k}}{\partial x^4} = \frac{1}{h_x^2} x \left[ \frac{\partial^2}{\partial x^2} (u(h+x)) + 2 \frac{\partial^2}{\partial x^2} (u(x)) + \frac{\partial^2}{\partial x^2} (u(x-h)) \right]$$

Il ne reste plus qu'à sommer nos termes et on obtient notre formule :

$$\frac{(\partial^4 u)_{i,j,k}}{\partial x^4} = \frac{1}{h_x^2} x \left[ \frac{\partial^2}{\partial x^2} (u(h+x)) + 2 \frac{\partial^2}{\partial x^2} (u(x)) + \frac{\partial^2}{\partial x^2} (u(x-h)) \right]$$

Ce qui nous donne la discrétisation suivante :

$$\frac{(\partial^4 u)_{i,j,k}}{\partial x^4} = \frac{u_{i+2,j,k} - 4u_{i+1,j,k} + 6u_{i,j,k} - 4u_{i-1,j,k} + u_{i-2,j,k}}{h_x^4}$$

Ensuite,

$$\frac{(\partial^4 u)_{i,j,k}}{\partial y^2 \partial x^2} = \frac{\partial^2}{\partial y^2} \left( \frac{\partial^2 u}{\partial x^2} \right)_{i,j,k} = \frac{1}{h_x^2} x \left[ \frac{\partial^2}{\partial y^2} (u(h+x)) + 2 \frac{\partial^2}{\partial y^2} (u(x)) + \frac{\partial^2}{\partial y^2} (u(x-h)) \right]$$

En faisant le même développement que lors de la premier dérivation on obtient :

$$\frac{(\partial^4 u)_{i,j,k}}{\partial x^2 \partial y^2} = \frac{u_{i+1,j+1,k} - 2u_{i+1,j,k} + u_{i+1,j-1,k} - 2u_{i,j+1,k} + 4u_{i,j,k} - 2u_{i,j-1,k} + u_{i-1,j+1,k} - 2u_{i-1,j,k} + u_{i-1,j-1,k}}{h_x^2 h_y^2}$$

Puis on va faire de même pour les autres opérateurs, ce qui va nous donner :

$$\frac{(\partial^4 u)_{i,j,k}}{\partial x^2 \partial z^2} = \frac{u_{i,j+1,k+1} - 2u_{i,j,k+1} + u_{i,j,k-1} - 2u_{i,j,k+1} + 4u_{i,j,k} - 2u_{i,j,k-1} + u_{i-1,j,k+1} - 2u_{i-1,j,k} + u_{i-1,j,k-1}}{h_x^2 h_z^2}$$

$$\frac{(\partial^4 u)_{i,j,k}}{\partial y^2 \partial z^2} = \frac{u_{i,j+1,k+1} - 2u_{i,j,k+1} + u_{i,j-1,k+1} - 2u_{i,j,k+1} + 4u_{i,j,k} - 2u_{i,j-1,k} + u_{i,j+1,k-1} - 2u_{i,j,k-1} + u_{i,j-1,k-1}}{h_y^2 h_z^2}$$

## 2.2 Discrétisation de nos problèmes :

Ici, je ne vais pas faire les développements, il seront écrits sur une feuille que je scanerai en annexe. Enfin la discrétisation des équations du problème 1 pour les points  $x=0$  et  $x = 1 - \frac{1}{h_x}$  nous donne :

Soit  $\Delta t$  le pas de discrétisation dans le temps :

$$\begin{cases} \frac{\partial u}{\partial t^2}(t, x, y, z) = \Delta^2(u)(t, x, y, z) - g(t, u(t, x, y, z)), \forall t \in [0, T], \forall (x, y, z) \in \Omega \\ u(0, x, y, z) = u_0(x, y, z), \forall (x, y, z) \in \Omega \\ u(t, x, y, z) = u(t, x+1, y, z), u(t, x, y, z) = u(t, x, y+1, z), \forall (x, y, z) \in \Omega \\ u(t, x, y, z) = u(t, x, y, z+1), \forall (x, y, z) \in \Omega \end{cases}$$

→

$$u_{ijk}^{n+1} - u_{ijk}^n - \Delta t \left( \frac{u_{i-1jk}^{n+1} - 2u_{ijk}^{n+1} + u_{i+1jk}^{n+1}}{h_x^2} + \frac{u_{ij-1k}^{n+1} - 2u_{ijk}^{n+1} + u_{ij+1k}^{n+1}}{h_y^2} + \frac{u_{ijk-1}^{n+1} - 2u_{ijk}^{n+1} + u_{ijk+1}^{n+1}}{h_z^2} + g(t^{n+1}, u_{ijk}^{n+1}) \right) = 0$$

Puis pour le problème 2, on commence par discrétiser

$$\Delta^2(u)(t, x, y, z)$$

→

$$\begin{aligned}
& \frac{u_{i+2jk}^{n+1} - 4u_{i+1jk}^{n+1} + 6u_{ijk}^{n+1} - 4u_{i-1jk}^{n+1} + u_{i-2jk}^{n+1}}{h_x^4} \\
& + \frac{u_{ij+2k}^{n+1} - 4u_{ij+1k}^{n+1} + 6u_{ijk}^{n+1} - 4u_{ij-1k}^{n+1} + u_{ij-2k}^{n+1}}{h_y^4} \\
& + \frac{u_{ijk+2}^{n+1} - 4u_{ijk+1}^{n+1} + 6u_{ijk}^{n+1} - 4u_{ijk-1}^{n+1} + u_{ijk-2}^{n+1}}{h_z^4} \\
& + \frac{u_{i+1j+1k}^{n+1} - 2u_{i+1jk}^{n+1} + u_{i+1j-1k}^{n+1} - 2u_{ij+1k}^{n+1} + 4u_{ijk}^{n+1} - 2u_{ij-1k}^{n+1} + u_{i-1j+1k}^{n+1} - 2u_{i-1jk}^{n+1} + u_{i-1j-1k}^{n+1}}{h_x^2 h_y^2} \\
& + \frac{u_{i+1jk+1}^{n+1} - 2u_{i+1jk}^{n+1} + u_{i+1jk-1}^{n+1} - 2u_{ijk+1}^{n+1} + 4u_{ijk}^{n+1} - 2u_{ijk-1}^{n+1} + u_{i-1jk+1}^{n+1} - 2u_{i-1jk}^{n+1} + u_{i-1jk-1}^{n+1}}{h_x^2 h_z^2} \\
& + \frac{u_{ij+1k+1}^{n+1} - 2u_{ij+1k}^{n+1} + u_{ij+1k-1}^{n+1} - 2u_{ij+1k}^{n+1} + 4u_{ijk}^{n+1} - 2u_{ij-1k}^{n+1} + u_{ij+1k-1}^{n+1} - 2u_{ijk-1}^{n+1} + u_{ij-1k-1}^{n+1}}{h_z^2 h_y^2}
\end{aligned}$$

Ensuite, on fait de même pour les autres termes de notre équations et on obtient :

$$\begin{cases}
\frac{\partial^2 u}{\partial t^2}(t, x, y, z) = \Delta^2(u)(t, x, y, z) + \Delta g(t, u(t, x, y, z)) + \frac{g(t, u(t, x, y, z))}{\partial t}, \forall t \in [0, T], \forall (x, y, z) \in \Omega \\
\frac{\partial u}{\partial t}(t, x, y, z) - \Delta(u)(t, x, y, z) - g(t, u(t, x, y, z)) = 0, \forall t \in [0, T], \forall (x, y, z) \in \Omega \\
u(0, x) = u_0(x), \forall (x, y, z) \in \Omega \\
u(t, x, y, z) = u(t, x+1, y, z), \forall (x, y, z) \in \Omega \\
u(t, x, y, z) = u(t, x+1, y, z), \forall (x, y, z) \in \Omega \\
u(t, x, y, z) = u(t, x, y+1, z), \forall (x, y, z) \in \Omega \\
u(t, x, y, z) = u(t, x, y, z+1), \forall (x, y, z) \in \Omega
\end{cases}
\longrightarrow$$

$$\begin{aligned}
& \Delta t^2 \left[ \frac{u_{i,j,k}^{n+1} - 2u_{i,j,k}^n + u_{i,j,k}^{n-1}}{h_x^2 h_y^2} \right. \\
& + \frac{u_{i+1,j+1,k} - 2u_{i+1,j,k} + u_{i+1,j-1,k} - 2u_{i,j+1,k} + 4u_{i,j,k} - 2u_{i,j-1,k} + u_{i-1,j+1,k} - 2u_{i-1,j,k} + u_{i-1,j-1,k}}{h_x^2 h_y^2} \\
& + \frac{u_{i+1,j,k+1} - 2u_{i+1,j,k} + u_{i+1,j,k-1} - 2u_{i,j,k+1} + 4u_{i,j,k} - 2u_{i,j,k-1} + u_{i-1,j,k+1} - 2u_{i-1,j,k} + u_{i-1,j,k-1}}{h_x^2 h_z^2} \\
& + \frac{u_{i,j+1,k+1} - 2u_{i,j,k+1} + u_{i,j-1,k+1} - 2u_{i,j+1,k} + 4u_{i,j,k} - 2u_{i,j-1,k} + u_{i,j+1,k-1} - 2u_{i,j,k-1} + u_{i,j-1,k-1}}{h_z^2 h_y^2} \\
& \left. + \frac{g(t^n+1, u_{i,j,k}^{n+1})}{\Delta t} \right] + \frac{g(t^n+1, u_{i,j,k}^{n+1}) - g(t^n, u_{i,j,k}^n)}{\Delta t} = 0
\end{aligned}$$

Maintenant que nous avons discrétisé nos problèmes, nous allons passer à l'implémentation en C++/PETSc.

### 3 Implementation PETSc :

#### 3.1 Présentation :

Dans une section précédente, nous avons présenté globalement ce qu'est la bibliothèque PETSc. Maintenant, nous allons commencer par une brève explication de comment utiliser cette bibliothèque :

Tout d'abord, l'initialisation de PETSc se fait de cette manière :

Listing 1 – Initialisation de PETSc

---

```

1  PetscErrorCode ierr ;
2  ierr = PetscInitialize(argc, argv, 0, 0) ;
3  CHKERRQ(ierr) ;
4  /* Utilisation de PETSc */
5  ierr = PetscFinalize() ; CHKERRQ(ierr) ;

```

---

#### 3.2 Présentation du travail qu'on va effectuer :

**Définition 1 (Maillage Dynamique)** *Le maillage dynamique est utilisé lorsqu'une frontière du domaine se déplace ou se déforme.*

##### 3.2.1 Intégration du problème 1 :

- ➡ On commencera par créer un maillage dynamique de taille  $N_x \times N_y \times N_z$  avec un stencil star de profondeur 1.
- ➡ Ensuite nous allons définir la fonction résidus SNES sous la forme locale
- ➡ Enfin, nous allons faire une étude sur le solveur optimal en terme de parallélisme.

##### 3.2.2 Intégration du problème 2 :

- ➡ On commencera par créer un maillage dynamique de taille  $N_t \times N_x \times N_y \times N_z$  avec un stencil box de profondeur 2.
- ➡ Ensuite nous allons définir la fonction résidus SNES sous la forme locale
- ➡ Enfin, nous allons faire une étude sur le solveur optimal en terme de parallélisme.

#### 3.3 Intégration du problème 1 :

On commence d'abord par définir une structure contenant nos paramètres physiques ;

---

```

1  #include <petscdmda.h>
2  #include <petscsnes.h>
3  typedef struct {
4      PetscReal dt;
5      PetscReal t;
6      PetscReal T;
7      PetscReal ihx2, ihy2, ihz2;
8      Vec v;
9      double (*g)(DMDALocalInfo*, int k, int j, int i, double, double);
10 } AppCtx;

```

---

**NB : PetscReal** permet de définir une variable  $\in \mathbb{R}$  tandis que **PetscInt** permet de définir une variable  $\in \mathbb{N}$

Après avoir fait cela on va définir une routine externe qui fera tout ce qu'il faut pour la partie 1 : Tout d'abord on initialise nos variables au sein de notre routine.

---

```

1 extern PetscErrorCode FormFunction(SNES, Vec, Vec, void *);
2 #undef __FUNCT__
3 #define __FUNCT__ "main"
4 int main(int argc, char **argv)
5 {
6     SNES          snes;
7     Vec           x;
8     AppCtx        user;
9     PetscErrorCode ierr;
10    DM             da;
11    DMDALocalInfo  info;
12    PetscInt       it, Nt=3;

```

---

### Initialisation de PETSc :




---

```
1 PetscInitialize(&argc,&argv,(char *)0,help);
```

---

### Explication :

Par l'intermédiaire de cette commande, on initialise la bibliothèque PETSc toute en Faisant un MPI\_Init. On va expliciter les argument de cette fonction :

-  **argc** : Compte le nombre d'arguments.
-  **args** : Permet de définir les arguments.
-  **help** : Permet de sortir le message d'aide (elle est optionnelle.).

### Création de notre maillage :

---

```

1 user.dt= 1e-3;
2 user.T=10.;
3 user.g=&gg;
4 ierr = PetscOptionsGetReal(NULL, "-dt",&user.dt,NULL);CHKERRQ(ierr);
5
6 ierr = SNESCreate(PETSC_COMM_WORLD,&snes);CHKERRQ(ierr);
7 ierr = DMDACreate3d(PETSC_COMM_WORLD,
8 DM_BOUNDARY_PERIODIC, DM_BOUNDARY_PERIODIC, DM_BOUNDARY_PERIODIC,
9 DMDA_STENCIL_STAR,-4,-4,-4,
10 PETSC_DECIDE,PETSC_DECIDE,PETSC_DECIDE,Nt,2,NULL,NULL,NULL,&da);CHKERRQ(ierr);
11
12 DMDAGetLocalInfo(da,&info);
13 user.ihx2=(info.mx-1)*(info.mx-1);
14 user.ihy2=(info.my-1)*(info.my-1);
15 user.ihz2=(info.mz-1)*(info.mz-1);
16
17 printf("info -> dof main:%d",info.dof);
18
19 ierr = DMSetApplicationContext(da,&user);CHKERRQ(ierr);

```

---

### Explication :

Ici on commence par définir nos paramètres de maillages  $\Delta t$ ,  $T$  et notre fonction  $g$ . Puis on utilise PetscOptionsGetReal pour obtenir la valeur de  $-\Delta t$  puis par le biais de CHKERRQ, on vérifie le code d'erreur. S'il n'est pas nul, il appelle le gestionnaire d'erreurs puis renvoie.

Ensuite cela étant fait, nous créons à la fois notre l'objet qui gèrera la communication de données de tableau 3d réparties sur notre maillage 3d et notre solveur d'équations non linéaires évolutifs par l'intermédiaire des mots clés DMDACreate3d et SNESCreate. Pour la première fonction, on utilise comme arguments : PETSC\_COMM\_WORLD pour dire qu'il faut mobiliser tous les processeurs, on définit un maillages périodiques selon les trois directions et le stencil à utiliser.



**Associer le da au snes**


---

```
1 ierr = SNESSetDM(snes, da);
```

---

**Associer la fonction FormFunctionLocal (de type DMDSNESFunction) aux da**


---

```
1 ierr = SNESSetDM(snes, da);
2 ierr = SNESSetFunction(snes, NULL, FormFunction, (void*)&user);
3 ierr = SNESSetFromOptions(snes);
```

---

**Notre seconde fonction :**

Ensuite, notre seconde fonction prend plusieurs argument, dont le solveur snes. Elle prend le solveur en argument pour nous laisser plus de flexibilité lors du choix des paramètres de définition.

---

```
1 PetscErrorCode FormFunction(SNES snes, Vec X,
2
3
4
5
6 PetscFunctionBeginUser;
7 ierr = SNESGetDM(snes, &da); CHKERRQ(ierr);
8 ierr = DMGetLocalInfo(da, &info);
9 ierr = DMGetLocalVector(da, &xlocalX); CHKERRQ(ierr);
10
11 ierr = DMGlobalToLocalBegin(da, X, INSERT_VALUES, xlocalX); CHKERRQ(ierr);
12 ierr = DMGlobalToLocalEnd(da, X, INSERT_VALUES, xlocalX); CHKERRQ(ierr);
13 ierr = DMDAVecGetArrayDOF(da, xlocalX, &arrayX); CHKERRQ(ierr);
```

---

**Explication :**

On commence par **PetscFunctionBeginUser**, elle doit être la première ligne exécutable de la routine PETSc fournie. Ensuite, on utilise **DMDAGetLocalInfo** pour obtenir des informations sur un tableau distribué donné et sur l'emplacement de ses processeurs.

On se sert ensuite de **DMGlobalToLocalBegin** pour mettre à jour le vecteur local à partir du vecteur global. La fonction **DMGlobalToLocalEnd** pour finir la mise à jour le vecteur local à partir du vecteur global. La fonction **DMDAVecGetArrayDOF** retourne un tableau à plusieurs dimensions qui partage des données avec le vecteur sous-jacent.

---

```
1 ierr = DMDAVecGetArrayDOF(da, F, &arrayF); CHKERRQ(ierr);
2 for (k=info.zs; k<info.zs+info.zm; k++) {
3   for (j=info.ys; j<info.ys+info.ym; j++) {
4     for (i=info.xs; i<info.xs+info.xm; i++) {
5       for (l=0; l<info.dof; l++) {
6         arrayF[k][j][i][l] = arrayX[k][j][i][l]-5.;}}}
7 ierr = DMDAVecRestoreArrayDOF(da, F, &arrayF); CHKERRQ(ierr);
8 ierr = DMDAVecRestoreArrayDOF(da, xlocalX, &arrayX);
9 ierr = DMRestoreLocalVector(da, &xlocalX);
10 FunctionNS3DVW: %f %f \n", norme2, normx);
11 ty \n");
12 PetscFunctionReturn(0);
13 }
```

---

**Explication :**

Ici on boucle selon nos trois directions pour parcourir tous les points du maillage. Puis on use de **DMDAVecRestoreArrayDOF** et **DMRestoreLocalVector** qui renvoie un vecteur Seq PETSc obtenu à partir de **DMGetLocalVector()** une fois que l'on a fini d'accéder aux entrées des vecteurs.

## 4 Conclusion :

Durant ce t.p., nous avons résolu le problème de l'équations de la chaleur tri-dimensionnelles et essayer de comparer les méthodes avec différents pré-conditionneurs et selon un nombre de processeurs différents.

Afin de parler du plan d'expérience, je tiens à préciser que PETSc est une bibliothèque bien plus puissante que MPI, malheureusement l'implementation en PETSc est beaucoup plus difficile.