Résolution parallèle de l'équation de la chaleur par éléments finis sur maillage partitionné

Projet AMS-TA01

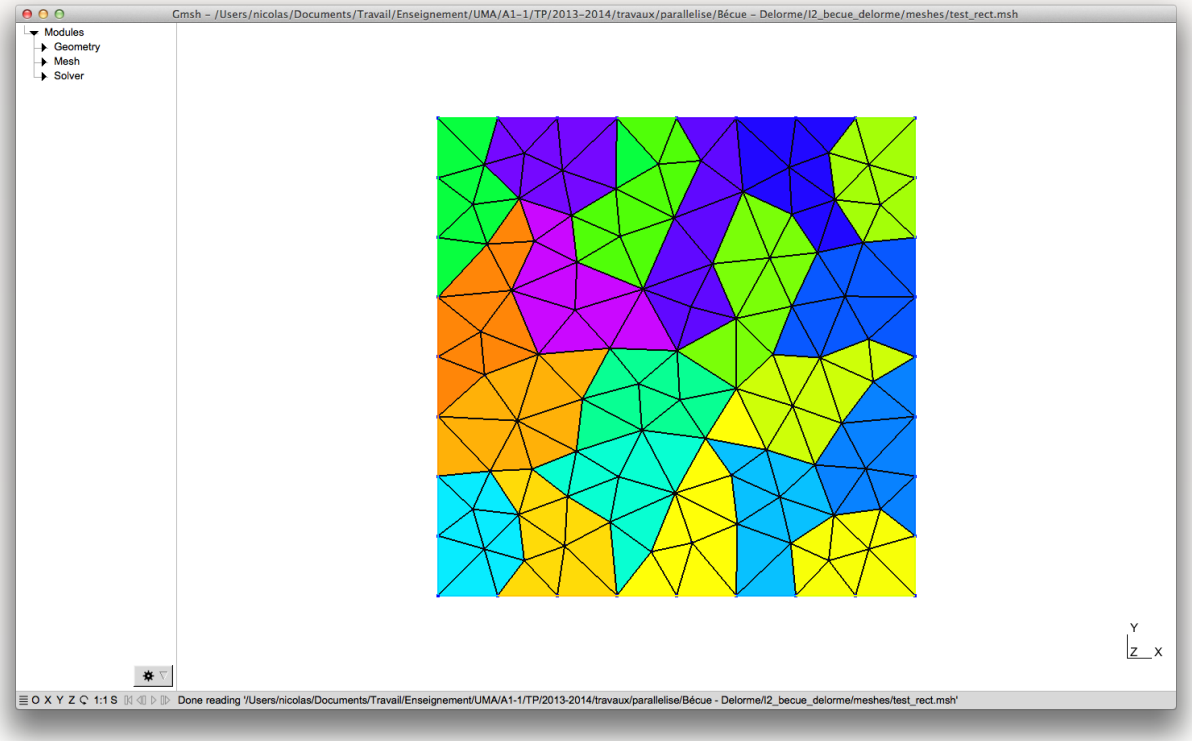
Nicolas KIELBASIEWICZ

POEMS, Unité de Mathématiques Appliquées, ENSTA - Paristech

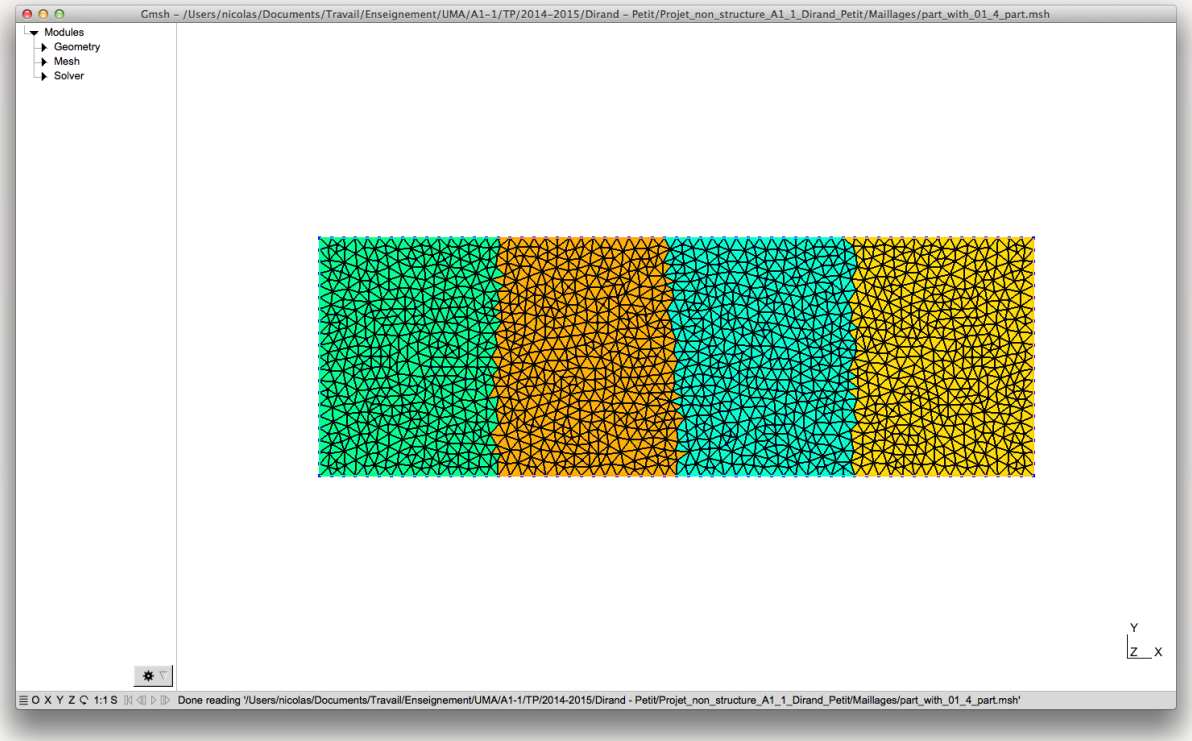
16 Oct 2015

Plan

- Résolution sur maillage non structuré partitionné



- Résolution sur maillage non structuré partitionné



- Résolution sur maillage non structuré partitionné
- Résolution par éléments finis de l'équation de la chaleur avec conditions aux limites de Dirichlet.

$$-\mathrm{div} k \nabla u = f \text{ on } \Omega \quad u = g \text{ in } d\Omega$$

- Résolution sur maillage non structuré partitionné
- Résolution par éléments finis de l'équation de la chaleur avec conditions aux limites de Dirichlet.

$$-\mathrm{div} k \nabla u = f \text{ on } \Omega \quad u = g \text{ in } d\Omega$$

- Implémentation de solveurs itératifs parallèles (Jacobi et Gauss-Seidel)

- Résolution sur maillage non structuré partitionné
- Résolution par éléments finis de l'équation de la chaleur avec conditions aux limites de Dirichlet.

$$-\mathrm{div} k \nabla u = f \text{ on } \Omega \quad u = g \text{ in } d\Omega$$

- Implémentation de solveurs itératifs parallèles (Jacobi et Gauss-Seidel)
- Etudes de scaling sur le code parallélisé

- Résolution sur maillage non structuré partitionné
- Résolution par éléments finis de l'équation de la chaleur avec conditions aux limites de Dirichlet.

$$-\mathrm{div} k \nabla u = f \ on \ \Omega \quad u = g \ in \ d\Omega$$

- Implémentation de solveurs itératifs parallèles (Jacobi et Gauss-Seidel)
- Etudes de scaling sur le code parallélisé

Elements de reflexion:

- Etapes de la parallélisation
- structure de données liées au partitionnement
- données à construire liées aux communications
- implémentation des solveurs Jacobi et Gauss-Seidel séquentiels

Les fichiers du code séquentiel sont téléchargeables ici:
`http://www.ensta-paristech.fr/~kielbasi/docs/AMS-TA01/`
Des tutoriels utiles pour la réalisation du projet: `http://www.ensta-paristech.fr/~kielbasi/?module=main&action=cours`

- Un tutoriel UNIX, au cas où
- Un tutoriel GMSH, pour générer vos propres maillages
- Un tutoriel FORTRAN 90, ça peut beaucoup ressembler à du matlab, mais pas toujours
- Un tutoriel MAKEFILE, au cas où

Plan

Le fichier amsta01sparse.f90 - type de données

Contient le module amsta01sparse permettant de manipuler des matrices creuses stockées au format sparse (comme matlab).

```
module amsta01sparse
  implicit none
  type matsparse
    integer :: n, m                                    ! nombre de lignes et nombre
        de colonnes de la matrice
    integer, dimension(:), pointer :: i, j             ! tableaux des indices de
        ligne et de colonne des coefficients non nuls
    real(kind=8), dimension(:), pointer :: val ! tableau des valeurs des
        coefficients
    logical :: isAllocated                             ! true si les 3 tableaux sont
        alloues (a la meme taille)
  end type
```

Le fichier amsta01sparse.f90 - opérateurs

- recherche de coefficients dans une matrice
- affectation dans une matrice
- Addition, soustraction, multiplication de matrice par un scalaire/vecteur/matrice

```
interface find
  module procedure findpos, findind
end interface
interface operator (+)
  module procedure spadd
end interface
interface operator (-)
    module procedure spminus
  end interface
  interface assignment (=)
  module procedure spaffect, spcopy
end interface
interface operator (*)
  module procedure spscalmat,spmatscal,spmatvec,spmatmat
end interface
```

Le fichier amsta01sparse.f90 - manipulation de matrice

sparse	déclaration de matrice
setcoeff	définition d'un coefficient
addtocoeff	ajout d'une valeur à un coefficient
multcoeff	multiplication d'un coefficient par une valeur
delcoeff	suppression d'un coefficient
msallocate	allocation mémoire d'une matrice
msdeallocate	désallocation mémoire d'une matrice
msreallocate	réallocation mémoire d'une matrice

Le fichier amsta01sparse.f90 - résolution d'un système linéaire

lufact	factorisation LU
lusolve	résolution d'un système factorisé LU

Le fichier amsta01maillage.f90 - type de données

Voir le tutoriel gmsh pour les détails du format

```
module amsta01maillage
  type maillage
    integer :: nbNodes, nbElems, nbTri
    real(kind=8), dimension(:,:), pointer :: coords
    integer, dimension(:,:), pointer :: typeElems, elemsVertices, triVertices
    integer, dimension(:), pointer :: refNodes, refElems, refTri,
        elemsNbPart, triNbPart, elemsPartRef, triPartRef
  end type
```

Le fichier amsta01maillage.f90 - fonctions disponibles

loadFromMshFile lecture d'un fichier de maillage GMSH

getTriangles construction des données sur les triangles

Le fichier amsta01probleme.f90 - type de données

```
type probleme
  type(maillage), pointer :: mesh
  real(kind=8), dimension(:), pointer :: uexa, g, u, f, felim
  type(matsparse) :: p_K, p_M, p_Kelim
end type
```

Le fichier amsta01probleme.f90 - fonctions disponibles

loadFromMesh	construit l'objet problème (le maillage n'est pas copié
kelem	calcul de la matrice de rigidité élémentaire
melem	calcul de la matrice de masse élémentaire
assemblage	assemblage des matrices éléments finis
pelim	pseudo-élimination des conditions essentielles
solveLU	résolution du système linéaire par factorisation LU
saveToVtu	sauvegarde des solutions calculée et exacte au format VTU

Le fichier main.f90

```
  ! lecture du maillage
  mail = loadFromMshFile("./testpart.msh")
  ! construction des donnees sur les triangles
  call getTriangles(mail)
  ! creation du probleme
  call loadFromMesh(pb,mail)
  ! assemblage des matrices elements finis
  call assemblage(pb)
  ! pseudo-elimination des conditions essentielles
  call pelim(pb,mail%refNodes(1))

  ! calcul du residu theorique
  allocate(residu(mail%nbNodes))
  residu=pb%felim-pb%p_Kelim*pb%uexa
  erreur=dsqrt(dot_product(residu,residu))
  print *, "residu theorique=", erreur

  ! resolution du systeme lineaire
  call solveLU(pb)

  ! calcul du residu
  residu=pb%felim-pb%p_Kelim*pb%u
  erreur=dsqrt(dot_product(residu,residu))
  print *, "residu=", erreur

  ! calcul de l'erreur L2
  erreur=dsqrt(dot_product(pb%uexa-pb%u,pb%uexa-pb%u))
  print *, "||u-uexa||_2=", erreur

  ! sauvegarde de la solution et de la solution theorique
  call saveToVtu(pb%mesh,pb%u,pb%uexa)
end program
```