UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

Rapport de Projet GLCS

Fait Par :
BOUYMEDJ Wissam
CISSE Aly

Groupe : M2 CHPS 2020/2021

Plan de travail

1 / **La première partie** : ……La déscription du fonctionnement du code fourni

2/ **La seconde partie** : ……...Ajout de fonctionnalités au code

2-1/ ……...Modification du système d'écriture des données [BOUYMEDJ Wissam]

2-2/…….Modification du système de configuration [CISSE Aly]

2-3/ ……Post-traitement en ligne des données [commun]

1/ La déscription du fonctionnement du code fourni :

Ce projet est focalisé sur la résolution d'une équation de la chaleur 2d en parallèle , en utilisant la programmation orientée objet c++, associée au communicateur MPI.

La matrice dans ce programme est divisé en domaine locale qui communique entre eux à l'aide d'MPI.
Description du code:

simpleheat.cpp:
construire la ligne de commande avec ses arguments, en utilisant la fonction config() de la classe CommandLineConfig. construire l'equation de la chaleur Ut+(i,j).

CartesianDistribution2D :
représente l'organisation d'un ensemble de processus sur une grille cartésienne (X,Y)
- Chaque processus a une coordonnée dans cette grille accessible avec la fonction coord ().
- Le nombre total de processus dans chaque dimension est fourni par la fonction extents ().
- Le communicateur est fourni par la fonction communicator () et sa taille ainsi que le rang du processus local sont disponibles via les fonctions size () et rank ().

CommandLineConfig.cpp:
la classe de configuration contient les arguments de configuration du programme: nombre d'iteration à excuter n_iter la forme du champ de données (2D) d

FinitediffHeatSolver.cpp : la classe qui construit le solveur de l'equation de la chaleur
la fonction iter(), elle permet de fixé les champs de la zone fontome de l'element next.

Une implémentation de 'TimeStep' qui Calcule le temps d'execution entre deux point consecutifs m_delta_t

FixedConditions.cpp : la classe qui fixe les conditions initiales

Simulation.cpp : La classe de simulation principale qui implémente le pas de temps.
cette classe a besoin des services des autres classes pour fournir la plupart de son comportement.

La simulation est observable par les instances de SimulationObserver grâce aux fonctionsobserve() et unobserve().

ScreenPrinter.cpp : Classe d'affichage (Une implémentation de Simulation Observer qui imprime les données de simulation sur l'écran)

Fonctionnement de la fonction :

Distributed2DField::sync_ghosts :

Cette fonction définie le sens dans lequel le processus courant voudrait transférer les données :
A travers sa variable locale nommée
up_randk =m_distribution.neighbour_rank(UP), elle fait appel la fonction

« neighbour_rank » qui elle même fait appel à la fonction MPI_Cart_shift, qui définie le rang des processus (sources, destination) selon le sens et la distance ; Ici cette variable locale transfère les données vers le processus d'en haut à une distance de 1 et dans le même sens que y (c-à-d de bas en haut). Inversement, la variable **« down_rank »** va transférer les données à l'inverse de la direction y.

Une fois ce paramétrage effectué, on appelle ensuite la fonction :

```
int MPI_Sendrecv(
  void *sendbuf,
  int sendcount,
  MPI_Datatype sendtype,
  int dest,
  int sendtag,
  void *recvbuf,
  int recvcount,
  MPI_Datatype recvtype,
  int source,
  int recvtag,
  MPI_Comm comm,
  MPI_Status *status
);
```

pour envoyer et recevoir des données suivant les modalités définis plus haut.

Avec :
sendbuf : désigne le pointeur vers notre données à envoyer
sendcount : la taille du tableau de données à envoyer
sendtype : type de données (type de variable dans MPI, ici c'est MPI_DOUBLE)
dest : left ou right
sendtag : ici COM_TAG
…
comm : désigne le communicateur et MPIStatus : liée à la reception

A travers l'appel à ces fonctions, on envoie les éléments internes à la limite de la zone courante, à la frontière de du processus juste en haut ou juste en bas. Qui constituera les éléments de la frontière du processus recepteur.

CartesianDistribution2D : cette classe identifie la fonction distribution du domaine local au sein du domaine global.

Elle contient deux données membres :
static constexpr int NDIM = 2 ;

qui définie la dimension dans laquelle nous voulons faire notre distribution de domaine.

Mpi_Comm m_comm ; Un communicateur mpi sur lequel plusieurs appels des fonctions MPI sera effectués.

Elle contient également plusieurs fonction membre dont :
Un constructeur qui prend en argument un nouveau communicateur MPI puis une variable indiquant le nombre de processus dans chaque direction et qui constructruit à travers le communicateur mpi donné en argument une topologie (grille 2d) sur notre communicateur défini en donnée membre.
Elle construit une grille 2d sur le communicateur de la donnée membre à travers la fonction :

```
int MPI_Cart_create(
  MPI_Comm comm,
  int ndims,
  int *dims,
  int *periods,
  int reorder,
  MPI_Comm *comm_cart
);

où :

comm : communicateur mpi

ndims : nombre de dimension de la grille cartésienne

periods : un tableau Boolean de taille ndims qui spécifie si la grille est
périodique ou non suivant chaque dimension.

Reorder : Indique si mpi peut organiser la grille
s'il le trouve nécessaire
```

2-1/ Modification du système d'écriture des données [BOUYMEDJ Wissam]

L'objectif de cette est de fournir une nouvelle classe pour remplacer l'écriture des données sur la console et stocke les données au format HDF5 .
Description du code:
on a crée une bibliothèque indépendante "**hdf5" ,** qui contient

- Un fichier **" hdf5.hpp "** écrits en c++, contient les prototypes des fonctions
- Un fichier **" hdf5.cpp "** écrits en c++, contient les corps des fonctions permettant de créer les files et stocker les données au format HDF5
- Compialtion

Un fichier CmakeLists.txt a été mis en place pour compiler

Pour le compiler, il suffit de taper :

```
>mkdir build
>cd mkdir
>cmake ..
```

- Description du code écrit dans “hdf5.cpp “ :
dans ce fichier on a implementé une class pour stocker data sous forme HDF5 FILE
pour créer des fichiers HDF5 on aura besoin de ces Attributs :

** nom de fichier
**la valeur qui est décrite par un datatype et dataspace

on a configuré un accés
m_fapl H5Pcreate(H5P_FILE_ACCESS)

apres on configure un accés parallel
m_fapl= H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(fapl MPI_COMM_WORLD,MPI_INFO_NULL)

et apres la creation de file

et o initialse à 0 un local pour compteur avec :
m_cu_iter = 0

et a chaque itération on fait appel pour simulation_updated pour faire une mis à jour pour les data

- et aussi on rajoute la class “hdf5” pour le fichier “simpleheat.cpp “

```
//create  the file and save the data
        hdf5(config, MPI_Comm comm);
        simulation.observe( hdf5);
```