Simulation des flux de chaleur dans un réacteur nucléaire
Rapport final

par
Stéphane Drouin

8 mai 2001

Introduction

Le but du projet est de simuler l'évolution des températures dans un réacteur nucléaire. En particulier, on devra considérer la température dans les barres de combustible et dans les canaux de caloporteur.

Par rapport à l'étape 2, le document de l'étape 3 apporte les modifications suivantes:

i) La description de la version finale de l'application a été ajoutée.

Analyse du problème

Barres de combustible

Soit la barre d'indices i et j. Chacune de ses faces est en présence d'un canal caloporteur que nous numéroterons ainsi:

	face 2	
face 1	BARRE	face 3
	face 4	

Dans chaque barre, on considère une section horizontale qu'on découpe en une grille de $n_g \times n_g$ points:

$i = 1$	.	.	.
$\vdots$	.	.	.
$i = n_g$	.	.	.
	$j = 1$	$\ldots$	$j = n_g$

Dans cette grille, la taille de chaque élément fini est $dx = l_b/n_g$. Pour chaque élément (i, j), et à chaque itération $n+1$ de $\Delta t(n)$ secondes, on calcule la nouvelle température:

$$T_{i,j}(n+1) = T_{i,j}(n) + \Delta T_{i,j}(n)$$

En utilisant

$$\Delta T_{i,j}(n) = \left(\frac{k}{\rho_b c_b}\nabla^2 T_{i,j}(n) + \frac{h}{\rho_b c_b}\right)\Delta t(n)$$

et les différences finies

$$\nabla^2 T_{i,j}(n) = \frac{T_{i,j+1}(n) + T_{i,j-1}(n) + T_{i+1,j}(n) + T_{i-1,j}(n) - 4T_{i,j}(n)}{dx^2}$$

Sur les bords de la grille, on a une situation particulière où un élément voisin n'est pas un point de la grille, mais directement le canal caloporteur. Dans ce cas, on a:

$$\begin{array}{rl} \text{Pour } i = 1 & T_{i-1,j}(n) = T_{face2}(n) \\ i = n_g & T_{i+1,j}(n) = T_{face4}(n) \\ \text{Et pour } j = 1 & T_{i,j-1}(n) = T_{face1}(n) \\ j = n_g & T_{i,j+1}(n) = T_{face3}(n) \end{array}$$

On calcule également le gradient de température sur chaque face:

$$\begin{aligned}
\nabla T_{face1}(n) &= \frac{\sum_{i=2}^{n_g-1}(T_{i,2}(n) - T_{i,1}(n))}{(n_g-2)dx} \\
\nabla T_{face2}(n) &= \frac{\sum_{j=2}^{n_g-1}(T_{2,j}(n) - T_{1,j}(n))}{(n_g-2)dx} \\
\nabla T_{face3}(n) &= \frac{\sum_{i=2}^{n_g-1}(T_{i,n_g-1}(n) - T_{i,n_g}(n))}{(n_g-2)dx} \\
\nabla T_{face4}(n) &= \frac{\sum_{j=2}^{n_g-1}(T_{n_g-1,j}(n) - T_{n_g,j}(n))}{(n_g-2)dx}
\end{aligned}$$

Canaux de caloporteur

Soit le canal d'indices i, j et k. Il est en présence de deux barres de combustible que nous numéroterons ainsi, selon k.
Pour $k = 1$:

barre 1	CANAL	barre 2

Pour $k = 2$:

barre 1
CANAL
barre 2

À chaque itération $n + 1$ de $\Delta t(n)$ secondes, on calcule la température moyenne du caloporteur:

$$T_c(n+1) = T_c(n) + \Delta T_c(n)$$

et

$$\Delta T_c(n) = \left(\frac{k}{c_c \rho_c l_c}(\nabla T_{f1}(n) + \nabla T_{f2}(n)) + \frac{2v}{L_b}(T_{in} - T_c(n))\right) \Delta t(n)$$

Sachant que, pour $k = 1$,

$$\begin{aligned}
\nabla T_{f1}(n) &= \nabla T_{barre1_{face3}}(n) \\
\nabla T_{f2}(n) &= \nabla T_{barre2_{face1}}(n)
\end{aligned}$$

et pour $k = 2$,

$$\begin{aligned}
\nabla T_{f1}(n) &= \nabla T_{barre1_{face4}}(n) \\
\nabla T_{f2}(n) &= \nabla T_{barre2_{face2}}(n)
\end{aligned}$$

De plus, on a, sur les bords du réacteur:

$$
\begin{aligned}
\text{Pour } j = 1,\ k = 1: &\quad \nabla T_{f1}(n) = 0 \\
\text{Pour } j = n_g + 1,\ k = 1: &\quad \nabla T_{f2}(n) = 0 \\
\text{Pour } i = 1,\ k = 2: &\quad \nabla T_{f1}(n) = 0 \\
\text{Pour } i = n_g + 1,\ k = 2: &\quad \nabla T_{f2}(n) = 0
\end{aligned}
$$

Stratégie de parallélisation

À chaque itération, le nombre de calculs est, pour chaque barre de combustible: $O(n_g^2)$ et, pour chaque canal caloporteur: $O(1)$.

Clairement, le calcul de la température des barres de combustible a intérêt à être distribué. Par contre, il serait inefficace de distribuer celui dans les canaux de caloporteur.

La stratégie à utiliser est la stratégie maître/esclave pour les barres de combustible. Un objet maître fournit à chaque objet barre de combustible esclave les valeurs de température dans les caloporteurs et lui demande de calculer la température dans les barres.

Pour les communications, 4 valeurs de température sont fournies à chaque barre et chaque barre est appelée à rendre 1 température maximale et 4 gradients de température.

Détermination des classes

Le problème suggère trois classes: **Barre** de combustible, **Canal** de caloporteur et **Reacteur**, qui assure les communications entre les tableaux de ces deux primitives et permet le contrôle de la simulation. De plus, les paramètres de la simulation sont placés dans une structure de données qui peut facilement être fournie aux composantes du système pour initialisation.

ParametresSimulation

Les paramètres sont placés dans la structure **ParametresSimulation** avec les variables suivantes:

Symb.	Param.	Description et valeur par défaut	Valeurs
v	cvit	Vitesse du caloporteur dans chaque canal: $4m/s$	$[0, \infty$
h	chin	Chaleur générée dans une barre: $5 \times 10^8 W/m^3$	$[0, \infty$
T_{in}	tempin	Température du caloporteur à l'entrée: $400K$	$[0, \infty$
k	diffusion	Diffusion de chaleur d'une barre: $2.2W/m/K$	$[0, \infty$
c_b	chspecb	Chaleur spécifique d'une barre: $220J/kg/K$	$]0, \infty$
c_c	chspecc	Chaleur spécifique du caloporteur: $698J/kg/K$	$]0, \infty$
ρ_b	densb	Densité d'une barre de combustible: $10^4 kg/m^3$	$]0, \infty$
ρ_c	densc	Densité du caloporteur: $800kg/m^3$	$]0, \infty$
L_b	longb	Longueur d'une barre de combustible: $0.5m$	$]0, \infty$
l_b	largb	Largeur d'une barre de combustible: $0.01m$	$]0, \infty$
l_c	largc	Largeur d'un canal de caloporteur: $0.02m$	$]0, \infty$
T_{fus}	Tfus	Température de fusion d'une barre: $2800K$	$[0, \infty$
n_g	ngrilles	Nombre de points dans les grilles: 10	≥ 3
n_b	nbarres	Nombre de barres dans le réacteur: 3	≥ 1

Barre

Pour une barre de combustible, on veut pouvoir obtenir la température maximale, obtenir le gradient de température pour toutes les faces, calculer une itération et initialiser les paramètres et placer le temps de simulation initial à 0.

Les méthodes suivantes sont donc définies pour une barre de combustible:

```
TempType temp_max();
TempArray grad_temp();
void Iteration(in TimeType pas, in TempArray temp_cal);
void Initialisation(in ParametresSimulation par);
```

Canal

Pour un canal de caloporteur, on veut pouvoir obtenir la température moyenne, calculer une itération et initialiser les paramètres et placer le temps de simulation initial à 0.

Les méthodes suivantes sont donc définies pour un canal de caloporteur:

```
TempType temp_moy();
void Iteration(in TimeType pas, in TempType grad_temp_1,
               in TempType grad_temp_2);
void Initialisation(in ParametresSimulation par);
```

Reacteur

Pour un réacteur, on veut pouvoir obtenir la température de la barre (i, j), obtenir la température du canal (i, j, k), calculer une itération et initialiser les paramètres de simulation et placer le temps de simulation initial à 0.

Les méthodes suivantes sont donc définies pour un réacteur:

```
TempType temp(in IndiceType i, in IndiceType j);
TempType calop(in IndiceType i, in IndiceType j, in IndiceType k);
void Iteration(in TimeType pas);
void Initialisation(in ParametresSimulation par);
```

Exceptions

On définit également les exceptions suivantes.

Si à une itération, une ou plusieurs barres de combustible ont atteint la température de fusion, le **Reacteur** lève une exception de type **Fusion** qui fournit le détail des barres en fusion.

De plus, si un calcul donne une température impossible (négative, par exemple) dans une **Barre** ou un **Canal**, une exception de type **ErreurCalcul** est levée. Notamment, on s'attend à ce que ce type d'erreur se produise en cas d'instabilité numérique.

À l'initialisation, si les paramètres fournis sont incompatibles avec les valeurs permises, l'objet initialisé lève une exception de type **MauvaisParametres** qui fournit un message d'erreur pertinent.

Toute demande d'accès à un élément inexistant du réacteur engendre une exception de type **MauvaisIndice** qui précise la nature de l'erreur.

Description de la version de base de l'application

Dans cette version, le programme fonctionne en mode statique: le nombre de barres de combustible est fixé à (2×2) barres. Le nombre de processeurs disponibles est fixé à 4: 1 client, 2 serveurs exécutant chacun 2 barres de combustible et 1 serveur exécutant 12 canaux de caloporteur.

Paramètres de simulation

Les paramètres de simulation sont écrits dans le fichier **config.txt** qui est situé dans le même répertoire que l'exécutable client. Chacun des paramètre peut y être précisé en inscrivant son nom (tel qu'indiqué dans la table des paramètres), suivi de la valeur désirée.

Un fichier de configuration typique est inclus en annexe.

Fonctionnement de l'application

L'IOR de chaque barre de combustible est écrit dans le fichier **barre.ref**. L'IOR de chaque canal de caloporteur est écrit dans le fichier **calop.ref**. Les serveurs sont responsables d'y écrire, à la fin des fichiers, ces IORs. Le client doit les lire et vérifier leur validité. Les IORs non-valides sont retirés des fichiers par le client. S'il n'y a pas suffissament d'IORs valides, le client signale l'erreur et termine.

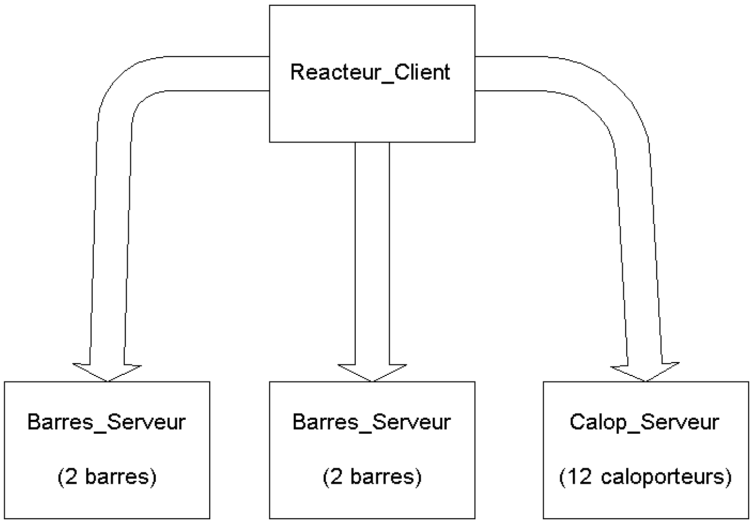

Figure 1: Fonctionnement de la version de base de l'application

Description de la version finale de l'application

Dans cette version, le programme fonctionne en mode dynamique: le nombre de barres de combustible peut être fixé à un nombre arbitraire. Le nombre de processeurs disponibles peut également être déterminé selon les besoins et n'a pas à être spécifié explicitement au client.

Paramètres de simulation

Comme à l'étape 2, les paramètres de simulation sont écrits dans le fichier **config.txt** qui est situé dans le même répertoire que l'exécutable client. Chacun des paramètre peut y être précisé en inscrivant son nom (tel qu'indiqué dans la table des paramètres), suivi de la valeur désirée.
Un fichier de configuration typique est inclus en annexe.

Fonctionnement de l'application

Chaque serveur offre une fabrique de canaux de combustible et une fabrique de canaux de caloporteur. L'IOR de chaque fabrique est écrit dans le serveur de noms. Le client doit les lire et vérifier leur validité. Les IORs non-valides sont retirés du serveur de noms par le client. S'il n'y a pas d'IOR valide pour les deux types de fabrique, le client signale l'erreur et termine. Le client est également responsable de répartir la charge de calcul sur tous les serveurs.

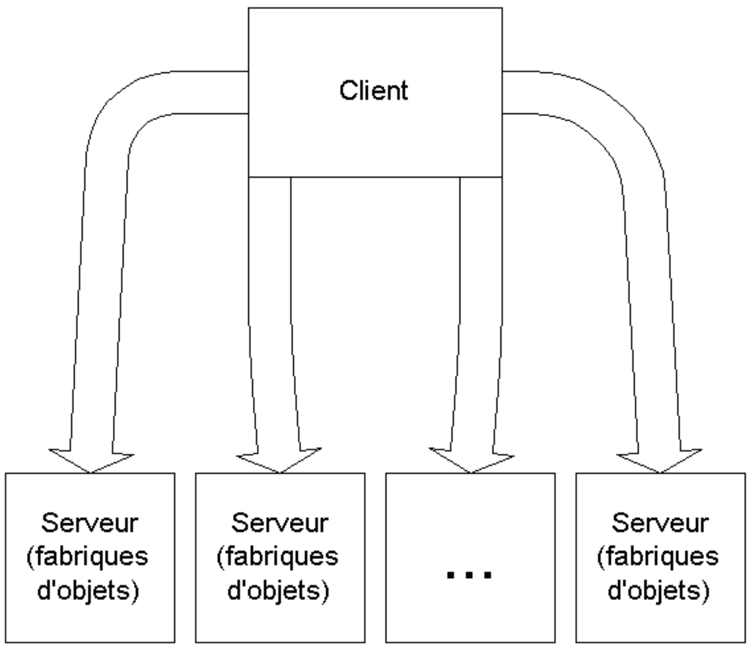

Figure 2: Fonctionnement de la version finale de l'application

Annexe A

```
// *********************************************************************
// reacteur.idl par Stephane Drouin
// Simulation des flux de chaleur dans un reacteur nucleaire
// Creation: 08/02/01 11:19:31
// Modification: 04/04/01: Ajout des fabriques d'objets
// *********************************************************************

#ifndef __REACTEUR_IDL__
#define __REACTEUR_IDL__

typedef double TempType;          // type pour les temperatures (kelvins)
typedef double TimeType;          // type pour le temps (en secondes)
typedef unsigned long IndiceType; // type pour les indices du reacteur
typedef double PhysicsType;       // type pour les dimensions physiques

struct ParametresSimulation {     // tous les parametres de simulation
  PhysicsType cvit;               // (modele mathematique du reacteur)
  TempType chin, tempin, diffusion, chspecb, chspecc;
  PhysicsType densb, densc;
  PhysicsType longb, largb, largc;
  TempType Tfus;
  IndiceType nbarres;             // taille du reacteur
  IndiceType ngrilles;            // nombre de points sur une barre
};

exception MauvaisParametres {     // si les parametres ne respectent pas
  ParametresSimulation par;       // ... les possibilites physiques
  string message;
};
exception ErreurCalcul {          // Erreurs possible de calcul;
  string message;                 // notamment l'instabilite numerique
};

interface Reacteur {              // Controle de la simulation
  void Initialisation(in ParametresSimulation par)
       raises(MauvaisParametres); // init variables et determine parametres
  typedef sequence<IndiceType> IndiceSeq;
  exception MauvaisIndice {       // demande d'acces a un item inexistant
    IndiceSeq indices;            // indice(s) de l'item demande
    string message;
  };
```

```
  TempType temp(in IndiceType i, in IndiceType j)
           raises(MauvaisIndice); // temperature maximale de la barre (i,j)
  TempType calop(in IndiceType i, in IndiceType j, in IndiceType k)
           raises(MauvaisIndice); // temp moyenne caloporteur (i,j,k)

  struct TempFusion {             // temperature de fusion atteinte!
    TempType temp;                // temperature atteinte
    IndiceType i,j;               // barre en cause
    TimeType instant;             // moment de la fusion
  };
  typedef sequence<TempFusion> TempFusionSeq;
  exception Fusion {              // a un instant, plusieurs barres peuvent
    TempFusionSeq details;        // ... atteindre la temperature de fusion
  };

  void Iteration(in TimeType pas) raises(Fusion);  // avancer la simulation
  TimeType TempsSimulation();     // Simulation depuis xx secondes
  IndiceType NombreDeBarres();    // Dimension du reacteur (un cote)
};

interface Barre {                 // Calcul de temperature dans une barre
  void Initialisation(in ParametresSimulation par)
       raises(MauvaisParametres); // init variables et fixe parametres
  void Detruire();                // le client detruit le serveur

  typedef TempType TempArray[4];  // 4 gradients de temperature
  TempArray grad_temp();          // gradient moyen sur les 4 faces
  TempType temp_max();            // temperature maximale de la barre
  void Iteration(in TimeType pas, in TempArray temp_cal)
       raises(ErreurCalcul);      // avancer simulation: calop voisins
};

interface Canal {                 // Calcul de la temperature dans un canal
  void Initialisation(in ParametresSimulation par)
       raises(MauvaisParametres); // init variables et fixe parametres
  void Detruire();                // le client detruit le serveur

  TempType temp_moy();            // temperature moyenne du caloporteur
  void Iteration(in TimeType pas, in TempType grad_1,in TempType grad_2)
       raises (ErreurCalcul);     // avancer simulation: gradients voisins
};
```

```
interface FabriqueDeBarres {      // creation dynamique de barres
  Barre creerObjet();             // creer un nouvel objet
};
typedef sequence<FabriqueDeBarres> desFabriqueDeBarres;

interface FabriqueDeCanaux {      // creation dynamique de canaux
  Canal creerObjet();             // creer un nouvel objet
};
typedef sequence<FabriqueDeCanaux> desFabriqueDeCanaux;

#endif
```

Annexe B

```
% Parametres de la simulation
% Simulation des flux de chaleur dans un reacteur nucleaire
%
% Les lignes commencant par '%' ou un espace sont ignorees. Les parametres
% qui ne sont pas modifies dans ce fichier prennent leur valeur par defaut.

% Vitesse du caloporteur dans chaque canal (def: 4 m/s)
cvit 4
% Chaleur generee dans une barre (def: 5e8 W/m3)
chin 5e8
% Temperature du caloporteur a l'entree (def: 400 K)
tempin 400

% Diffusion de chaleur dans une barre (def: 2.2 W/m/K)
diffusion 2.2
% Chaleur specifique d'une barre (def: 220 J/kg/K)
chspecb 220
% Chaleur specifique du caloporteur (def: 698 J/kg/K)
chspecc 698
% Densite d'une barre de combustible (def: 1e4 kg/m3)
densb 1e4
% Densite du caloporteur (def: 800 kg/m3)
densc 800

% Longueur d'une barre de combustible (def: 0.5 m)
longb 0.5
% Largeur d'une barre de combustible (def: 0.01 m)
largb 0.01
% Largeur d'un canal de caloporteur (def: 0.02 m)
largc 0.02

% Temperature de fusion d'une barre (def: 2800 K)
Tfus 2800

% Nombre de points dans une grille modelisant une barre (def: 10)
ngrilles 10
% Nombre de barres dans le reacteur (def: 3)
nbarres 4
```

Annexe C

```
Explications pour faire fonctionner les executables de maniere
distribuee, sur un ou plusieurs ordinateurs.

L'application est composee de

-deux fichiers executables (Client.exe et Serveur.exe);
-un fichier de configurations (Config.txt);

Notes generales

-Il doit y avoir au moins un Serveur qui s'execute avant de lancer Client.
-En tout temps, le fichier de configuration (Config.txt) doit etre
 place dans le repertoire courant lors de l'execution de Client.
 Si ce fichier n'existe pas, les parametres par defaut sont utilises.

Les etapes suivantes doivent etre respectees pour executer l'application
------------------------------------------------------------------------
1) Executer un serveur de noms. (gonameserv.bat peut etre utilise)
2) Executer autant de fois que desire le Serveur.exe en specifiant
   le serveur de noms (goserveur.bat peut etre utilise)
3) Executer Client.exe en specifiant le serveur de noms (goclient.bat peut
   etre utilise)
```

Annexe D

```
Explications pour reproduire les executables. L'application est composee de
trois projets: contrat IDL, client et serveur

Projet Reacteur
---------------
Le fichier suivant est compile
 - Reacteur.idl             (Contrat IDL)

Pour produire les fichiers
 - Reacteur.cpp
 - Reacteur.h
 - Reacteur_skel.cpp
 - Reacteur_skel.h

Projet Client
-------------
Les fichiers suivants, tous situes dans le repertoire principal du projet,
sont compiles
 - std_main.cpp             (initialisation standard du bus CORBA)
 - Client_run.cpp           (procedure principale du programme)
 - Reacteur.cpp             (produit par le compilateur IDL)
 - Reacteur_impl.cpp        (implantation du reacteur (non distribue))

Pour produire, comme resultat final
 - Client.exe

Projet Serveur
--------------
Les fichiers suivants, tous situes dans le repertoire principal du projet,
sont compiles
 - std_main.cpp             (initialisation standard du bus CORBA)
 - Serveur_run.cpp          (procedure principale du programme)
 - Reacteur.cpp             (produit par le compilateur IDL)
 - Reacteur_skel.cpp        (produit par le compilateur IDL)
 - Barre_impl.cpp           (implantation d'une barre (distribue))
 - Canal_impl.cpp           (implantation d'un caloporteur (distribue))
 - FabriqueDeBarres_impl.cpp (implantation d'une fabrique de barres)
 - FabriqueDeCanaux_impl.cpp (implantation d'une fabrique de caloporteurs)

Pour produire, comme resultat final
 - Serveur.exe
```