Transfert de chaleur

Programmation concurrente

Etienne STROBBE

Contexte

Dans ce projet, nous souhaitons programmer un simulateur qui utilise un modèle très simple du phénomène de transfert de chaleur par conduction. Nous voulons calculer l'évolution de la température sur la face interne d'une cloison en fonction de l'évolution de la température sur sa face externe.

Mise en œuvre

Implémentation

L'implémentation de cette simulation a été faite entièrement en Java.

Le mur est représenté par une matrice globale de taille n*m (n itérations, m tranches du mur, qui permet uniquement un affichage de toutes les valeurs).

Un tableau de taille m (m tranches) contenant les constantes est également présent. Dans ce tableau, la constante à l'indice i correspond à la constante du mur à l'indice i (en rapport au premier tableau contenant les températures).

La création de ce deuxième tableau permet de simplifier le calcul des températures.

L'ensemble des calculs pour la simulation se déroule dans la classe Simulateur.

Pour cette troisième version de la simulation, le calcul est effectué en parallèle par différents Threads java.

Le mur est séparé en 9 tranches, dont 2 où les valeurs restent fixent (les deux tranches extrêmes).

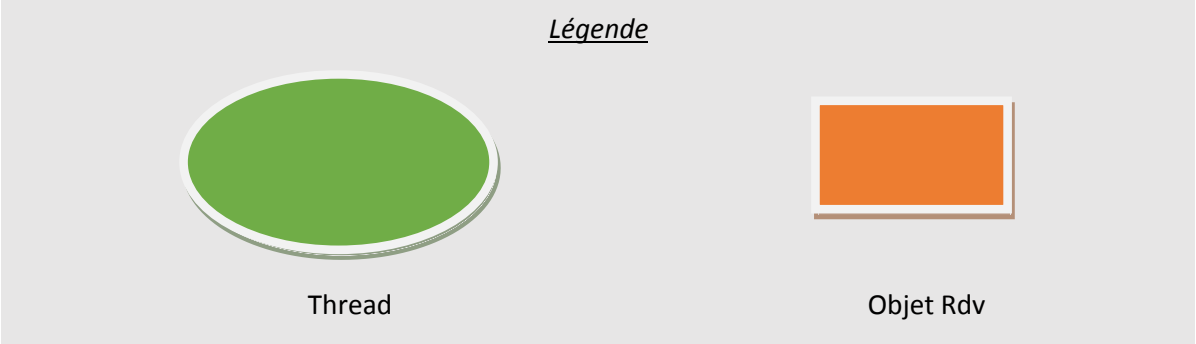
Lors du lancement de la simulation, on fait appel à la méthode 'simule()' qui effectue les actions suivantes :

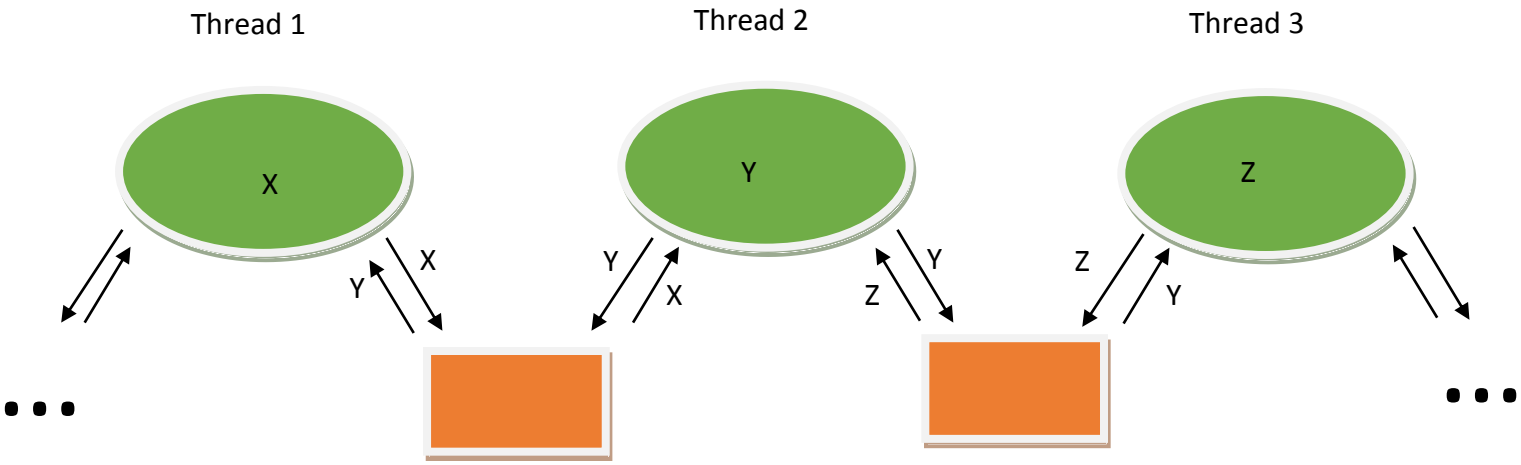
- On enregistre la date actuelle pour calculer le temps d'exécution
- 7 thread sont créés (1 pour chaque tranche dont la température varie durant la simulation) et lancés
- On attend la fin de la simulation à l'aide d'une variable globale
- On enregistre la date actuelle pour calculer le temps d'exécution

```
void simule() {
        debut = date actuelle
        creer thread(7)
        Tant que (pas fini){// fini quand tous les threads ont terminé
                Attendre 5ms
        }
        fin = date actuelle
}
```

La synchronisation entre les Threads se fait à l'aide de rendez-vous.

Schéma explicatif :





Chaque thread représentant une tranche, et donc chaque thread prend 2 rendez-vous, un avec le thread représentant la tranche suivante et un autre avec le thread représentant la tranche précédente.

Les rendez-vous sont effectués à l'aide des objets Rdv (chaque Rdv effectue un rendez-vous entre 2 threads).

Quand un thread prend un rendez-vous, il donne à l'objet Rdv la dernière température connue de la tranche qu'il représente et l'objet Rdv lui renvoi la température donné par le thread avec lequel il a pris rendez-vous (il y a échange de température entre les deux threads).

Dans le schéma ci-dessus, on peut voir que le Thread 2 échange sa température Y avec le Thread 1 et reçoit la température X, de même, sur le deuxième rendez-vous, il échange sa température Y avec le Thread 3 qui lui donne la température Z.

Lors de la création des différents threads, on associe à chaque thread les deux objets Rdv de façon similaire au schéma ci-dessus.

```
void creer thread (entier nombre) {
        Rdv premier
        Rdv second
        pour (entier i=1, jusqu'à i=taille du mur-1)
                //on ne crée pas de threads pour les deux tranches extrêmes
                Créer ThreadSimulation(premier , second)
                Lancer ThreadSimulation
                premier = second
                second = nouveau Rdv
        Fin pour
}
```

ThreadSimulation est un objet qui hérite de la classe Thread en java.

La méthode exécutée au lancement de chacun de ces threads et la suivante :

```
void run () {
        //on suppose que tmp est la temperature actuelle de la tranche représentée
        //tmpAvant et tmpApres représentent respectivement la température
        // de la tranche d'avant et la température de la tranche d'après
        // gauche est l'objet Rdv de gauche et droite est l'objet Rdv de droite
        pour (entier it=0, jusqu'à it=dernière iteration)
                tmpAvant = gauche.récupérer valeur de gauche (tmp)
                tmpApres = droite.récupérer valeur de droite (tmp)
                tmp = f(tmp,tmpAvant,tmpApres)
                // la fonction f calcul la nouvelle valeur de tmp
        fin pour
        Thread terminé //modifie la variable globale
}
```

Dans la fonction f qui calcul la nouvelle température de la tranche, on insère cette température dans la matrice globale.

L'implémentation de la classe Rdv se fait simplement comme suit :

```
Class Rdv {
        flottant alpha,beta // variables pour faire l'échange
        entier attente // le nombre de thread en attente

        echange(flottant tmp){
                si(attente = 0)
                        attente = 1
                        alpha = tmp
                        attendre
                        renvoyer beta // échange de valeur
                sinon
                        attente = 0
                        beta = tmp
                        notifier
                        renvoyer alpha // échange de valeur
                fin si
        }
}
```

Calcul des constantes

Avec la formule suivante :

$$\mathbf{T(x,t+\varepsilon) = T(x,t) + \frac{\lambda\varepsilon}{\rho c}\left(\frac{T(x+\delta)+T(x-\delta)-2T(x)}{\delta^2}\right)}$$

Avec $\varepsilon = 1 : T(x,t+1) = T(x,t) + C * \big(T(x+1,t) + T(x-1,t) - 2T(x,t)\big)$

On en déduit aisément donc $C = \frac{\lambda}{\rho c \times \delta^2}$

Avec un pas de temps de 600s et un pas d'espace de 0.04cm on trouve les valeurs suivantes :

Coefficient brique : 0.26785714285714285

Coefficient laine de verre : 0.5555555555555555

Au début de la simulation, ces constantes sont calculées (avant même la création des threads).

Avec un pas de temps de 1 seconde, il nous faudrait 31 546 000 (365*24*3600) itérations pour simuler une année entière.

Résultats

Détaillés

Voici les valeurs des différentes valeurs des températures du mur toutes les heures pendant 10 heures pour un pas de temps de 600s et un pas d'espace de 0.04cm :

```
t=0h ->[110,20,20,20,20,20 - 20,20,20,20 ]
t=1h ->[110,73,46,29,22,20 - 20,20,20,20 ]
t=2h ->[110,83,60,42,30,23 - 23,21,20,20 ]
t=3h ->[110,88,67,50,37,27 - 27,23,21,20 ]
t=4h ->[110,90,72,55,41,29 - 29,24,22,20 ]
t=5h ->[110,92,74,58,44,31 - 31,25,22,20 ]
t=6h ->[110,93,76,61,46,32 - 32,26,23,20 ]
t=7h ->[110,94,77,62,47,33 - 33,26,23,20 ]
t=8h ->[110,94,78,63,48,33 - 33,26,23,20 ]
t=9h ->[110,94,79,63,48,34 - 34,27,23,20 ]
t=10h ->[110,95,79,64,49,34 - 34,27,23,20 ]
```

Temps

Nous avons observé une moyenne de temps d'exécution de 1000 ms (moyenne effectuée sur une série de 10 simulations) pour la simulation de 100.000 cycles de simulation.

Ce temps d'exécution a été calculé grâce à la classe Date en Java. Comme décrit dans l'implémentation, la méthode 'simule()' enregistre la date de début et la date de fin, ainsi, en faisant la différence des deux dates, on obtient le temps d'exécution.

Changements notables

On peut apercevoir sur la sortie que la face interne du mur (celle qui ne reçoit pas le soleil), voit sa température changer (1°C de différence) à partir de 2,8 heures (10200 secondes) de simulations.