Transfert de chaleur V4

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 C Posix.

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).

Plusieurs autres variables globales sont utilisées dans cette implémentation : deux pour les constantes C1 et C2 et deux pour les pas de temps et de distance (DT et DX).

L'ensemble des calculs pour la simulation se déroule dans le fichier main.c.

Pour cette quatrième version de la simulation, le calcul est effectué en parallèle par différents Threads en C Posix.

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

Lors du lancement de la simulation, la méthode main() est appelé, elle effectue les actions suivantes :

- On initialise les constantes pour le calcul
- 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 en faisant un join
- On enregistre la date actuelle pour calculer le temps d'exécution
- On affiche la matrice globale pour afficher l'évolution des calculs

```
main() {
        initialization des constantes
        debut = date actuelle
        creation des threads // détaillé plus bas (voir Création des threads)
        join les threads
        fin = date actuelle
        afficher matrice globale
}
```

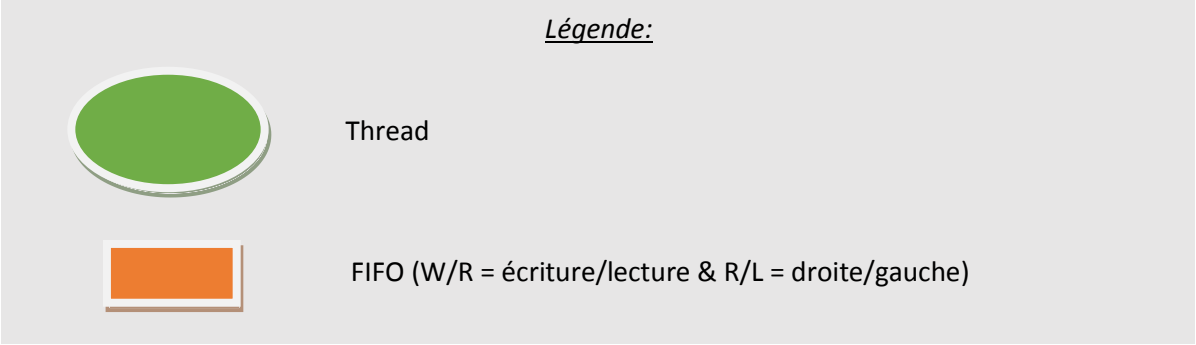
Description des FIFOs dans les threads.

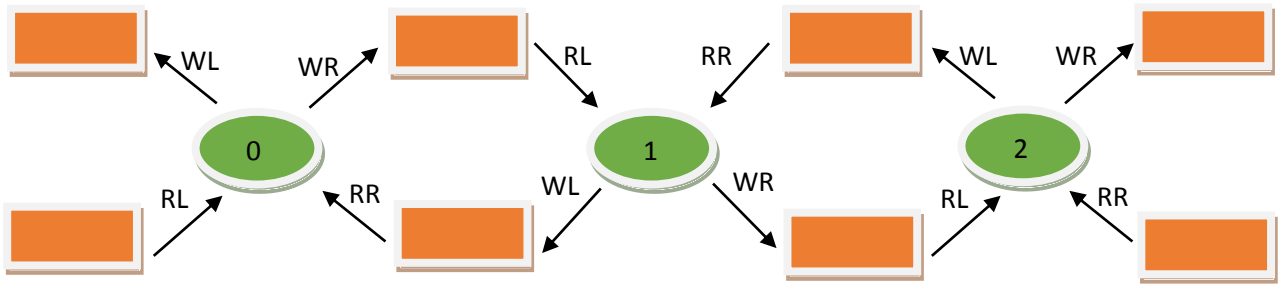
La synchronisation entre les Threads se fait à l'aide de FIFOs de taille 1 (on reproduit le modèle producteur-consommateur)

Chacune des threads a accès à quatre FIFOs, deux en lecture et deux en écriture. Deux FIFOs de chaque thread sont partagés avec les threads adjacents (voir schéma). Lors du calcul de la nouvelle température à l'instant t, le thread a besoin des valeurs en (x), (x-1) et (x+1) à l'instant (t-1).

Pour la valeur en (x) à l'instant (t-1), le thread possède déjà cette valeur en local, pour les deux autres valeurs, il va les récupérer dans les deux FIFOs en lecture, celui de gauche pour la valeur en (x-1) et celui de droite pour la valeur en (x+1). Une fois le calcul effectué, le thread ajoute la nouvelle valeur calculée dans les deux FIFOs en écriture afin de partage cette valeur avec les threads adjacents.

Schéma explicatif :





Comme on peut le voir sur le schéma ci-dessus, le thread n°1 partage les FIFO WR et RR avec le thread n°2 (qui deviennent respectivement RL et WL pour le thread n°2) et de même pour le thread n°0 (WL et RL du thread n°1 deviennent respectivement RR et WR du thread n°0).

La synchronisation entre les différents threads est possible grâce à deux points importants :

- La taille des FIFOs est de 1, et donc il n'est possible d'ajouter un élément dans une FIFO que si celle-ci est vide (de même, il n'est possible de prendre une valeur dans la FIFO que si celle-ci est pleine). Voir *Description des opérations sur les FIFOs.*
- Les deux threads représentant les tranches extrêmes du mur ne possèdent que deux FIFOs.
    - le thread de gauche ne possède que les FIFOs WR et LR, en effet il n'a pas besoin de partager sa valeur avec le thread de gauche puisque qu'il est le premier thread et il n'a également pas besoin de récupérer la valeur du thread de gauche puisque cette valeur est fixe et vaut 110.
    - de même pour le thread de droite, il ne possède que les FIFOs WL et RL (pour les mêmes raisons)

Création des threads.

Comme décrit précédemment, les threads sont créés et lancés dans la méthode main et il existe trois types de threads différents : celui de l'extrême gauche, celui de l'extrême droite et ceux entre les deux.

Pour utiliser les threads en C Posix, on a utilisé ici pthread. On passe en paramètre (à la création/lancement du thread) une structure en C, comportant quatre FIFOs. On définit différemment ces structures selon le type de thread.

Voici le pseudo code correspondant à l'initialisation/lancement de ces différents threads :

```
Créer structure (paramètre) avec 2 nouvelles fifos (écriture-droite et lecture-droite)
Conserver la valeur des fifos droite
Lancer un thread le paramètre et méthode de calcul gauche // voir Calcul des threads
pour (entier i=1, jusqu'à i=5)
        Créer structure (paramètre) avec 2 nouvelles fifos (écriture-droite et lecture-droite)
        Ajouter 2 fifos à cette structure avec les valeurs conservées
        //écriture-gauche=ancienne lecture-droite & lecture-gauche=ancienne écriture-droite
        Lancer un thread avec le paramètre et méthode de calcul générale
        Conserver la valeur des fifos droite
Fin pour
Créer une structure (paramètre) vide
Ajouter 2 fifos à cette structure avec les valeurs conservées
//écriture-gauche=ancienne lecture-droite & lecture-gauche=ancienne écriture-droite
Lancer un thread avec le paramètre et méthode de calcul droite
```

Calcul des threads.

Il y a trois types de threads différents et pour chacun, une méthode de calcul différente (pour assurer la synchronisation et prévenir les interblocages).

Voici les trois différentes méthodes de calculs pour les trois différents types de threads :

1. Méthode de calcul générale :

```
void calcul_général() {
        //on suppose que valeur est la temperature actuelle de la tranche représentée
        //lecture-gauche et lecture-droite sont les fifos de lecture du thread courant
        //écriture-gauche et écriture-droite sont les fifos d'écriture du thread courant
        pour (entier it=0, jusqu'à it=dernière iteration)
                valeur = f(valeur, valeur de lecture-gauche, valeur de lecture-droite)
                // la fonction f calcul la nouvelle valeur de valeur
                Ajouter valeur dans écriture-gauche et écriture-droite
        fin pour
}
```

2. Méthode de calcul gauche :

```
void calcul_gauche() {
        //on suppose que valeur est la temperature actuelle de la tranche représentée
        //lecture-droite est la fifo de lecture du thread courant
        //écriture-droite est la fifo d'écriture du thread courant
        pour (entier it=0, jusqu'à it=dernière iteration)
                valeur = f(valeur, 110.0, valeur de lecture-droite)
                // la fonction f calcul la nouvelle valeur de valeur
                Ajouter valeur dans écriture-droite
        fin pour
}
```

3. Méthode de calcul droite

```
void calcul_droite() {
        //on suppose que valeur est la temperature actuelle de la tranche représentée
        //lecture-gauche est la fifo de lecture du thread courant
        //écriture-gauche est la fifo d'écriture du thread courant
        pour (entier it=0, jusqu'à it=dernière iteration)
                valeur = f(valeur, valeur de lecture-gauche, 20.0)
                // la fonction f calcul la nouvelle valeur de valeur
                Ajouter valeur dans écriture-gauche
        fin pour
}
```

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

Dans le calcul de la nouvelle valeur, comme dans les versions précédentes, on différencie les cas selon la position de la tranche que l'on calcul (les coefficients changent).

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 -> [110,74,47,31,23,21 - 21,20,20,20 ]
t=2h -> [110,83,60,43,31,25 - 25,23,21,20 ]
t=3h -> [110,87,67,51,38,29 - 29,26,22,20 ]
t=4h -> [110,90,72,56,43,32 - 32,28,24,20 ]
t=5h -> [110,92,75,60,46,35 - 35,29,24,20 ]
t=6h -> [110,93,77,62,49,36 - 36,31,25,20 ]
t=7h -> [110,94,79,64,50,37 - 37,31,25,20 ]
t=8h -> [110,94,80,65,51,38 - 38,32,26,20 ]
t=9h -> [110,95,80,66,52,38 - 38,32,26,20 ]
t=10h ->[110,95,81,66,53,39 - 39,32,26,20 ]
```

Temps

Pour calculer le temps d'exécution, on a utilisé ici deux variables de type clock_t (une initialisée au début de la simulation et une initialisée à la fin) et on a fait la différence entre les deux afin de connaître le temps d'exécution.

Nous avons observé une moyenne de temps d'exécution de 3720 ms (moyenne effectuée sur une série de 10 simulations) pour la simulation de 100.000 cycles de simulation.

La version itérative prend, quant à elle, 20ms pour s'exécuter. Cette différence est due au faite que les calculs sont trop simples et que la méthode de synchronisation est un peu lourde pour ce genre de calcul (l'implémentation des FIFOs pourrait être également simplifiée), mais ici on ne s'intéresse pas à la vitesse d'exécution mais à l'apprentissage des méthodes de synchronisation des calculs exécutés en parallèle.

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 2,333 heures de simulations.