AMS-I03 TP2
Modèles et techniques en programmation parallèle hybride et multi-cœurs

Dongshu LIU

1 Rappelle du problème

Chercher une solution $u : (x,t) - - > u(x,t)$, où $x \in \Omega = [0,1]^3$ et $t \geq 0$ qui vérifie :

$$\frac{\partial u}{\partial t} = \nabla u + f(x,t) \tag{1}$$

$$u(x,0) = g(x) \qquad x \in \Omega \tag{2}$$

$$u(x,t) = g(x) \qquad x \in \partial\Omega, t > 0 \tag{3}$$

où f et g sont des fonctions donées.

2 Les résultats en version MPI

On lance les codes de calcul en version MPI, on peut recevoir :
Pour $n_1 = 800, n_2 = n_3 = 400$, $it = 10$

```
1 processus
temps init    5.59853 s  temps total  195  s

2 processus
temps init    4.5525  s  temps total  97.6 s

4 processus
temps init    4.52043 s  temps total  55.4 s
```

3 Version MPI-OpenMP (grain fin)

Pour la methode de grain fin, j'ai modifié la partie de "scheme.cxx" et "values.cxx" en ajoutant "pragma omp parallel for" autour de la boucle en domaine.

Les résultats sont suivant ($n_1 = 800, n_2 = n_3 = 400$, $it = 10$) :

```
1 processus x 8 threads
temps init    5.6486  s  temps total  114  s

2 processus x 4 threads
```

```
temps init    4.63004 s  temps total  62.8 s

4 processus x 2 threads
temps init    4.54602 s  temps total  47.1 s
```

On peut voir que généralement version de multi-threads est plus rapide que version MPI. Et pour le meme nombre threads, quand le nombre de processus est plus grand, le vitesse est plus grand.

4 Version MPI-OpenMP (grain grossier)

Pour la methode de grain grossier, j'ai ajouté la partie de parallèle autour le boucle de temps, et modifié "schme.cxx" et "values.cxx" .

Les résultats sont suivant ($n_1 = 800, n_2 = n_3 = 400$, $it = 10$) :

```
1 processus x 8 threads
temps init    3.01912 s  temps total  107  s

2 processus x 4 threads
temps init    2.63757 s  temps total  59.4 s

4 processus x 2 threads
temps init    3.10427 s  temps total  45.3 s
```

On voit que cette fois l'accélération d'initiation est plus évident, et on peut aussi distinguer l'avantage de version "grain grossier" comparée avec la version "grain fin" : le temps total a diminué 10% environ.

L'algorithme de découpage domaine MPI en sous-domaines

Dans "parameters.cxx" lignes 143-169 :

D'abord, les matrices "$m_i min_t hread$" et "$m_i max_t hread$" (taille 3 direction x nombre de threads) initialisent avec les valeurs de maximum et minimum dans la domine MPI.

En suite, il calcule le nombre de point dans chaque sous-domaine, presenté par"di".

Finalement, dans chaque direction, récalcule les valeurs minimum et maximum de chaque sous-domaine (maximum = minimum + di ; minimum de sous-domaine suivant = maximum de sousdomaine précédente + 1) ;. Le variable 'idecoupe' est pour aussuer qu'on fait la decouplage que dans la direction dont le data est plus grand.

Si jamais le maximum de la dernière sous-domaine est plus grand, on le définit avec le maximum de domaine MPI.