AMS-I03 TP1

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 de code séquentiel

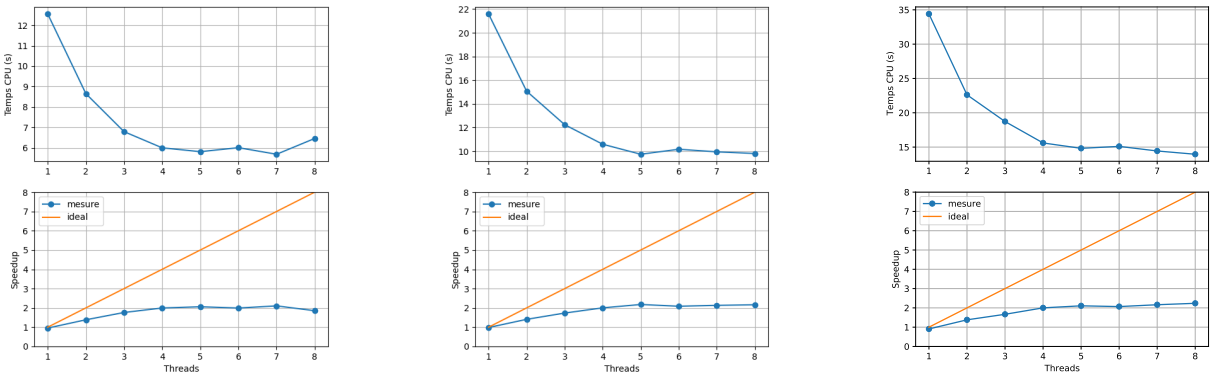
On lance les codes de calcul séquentiel, on peut recevoir :
Pour $n_1 = n_2 = n_3 = 400$, $it = 10$, $dt = 6.25 * e^{-07}$

```
 temps init    0.95826 s

iteration     0  variation    59709.3  temps calcul    1.08717 s
iteration     1  variation    33911.6  temps calcul    2.15817 s
iteration     2  variation    25175.9  temps calcul    3.22384 s
iteration     3  variation    20512.3  temps calcul    4.29167 s
iteration     4  variation    17677.6  temps calcul    5.36896 s
iteration     5  variation    15751.3  temps calcul    6.46631 s
iteration     6  variation    14346.4  temps calcul    7.61658 s
iteration     7  variation    13266.9  temps calcul    8.79095 s
iteration     8  variation    12404.8  temps calcul    10.0299 s
iteration     9  variation    11695.5  temps calcul    11.1718 s

terminate PoissonSeq

               temps total   12.1306 s
```

3 Version multithreads avec 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.

Après j'ai lancé la calcul sur 3 domaine différentes : $[0, 399] \times [0, 399] \times [0, 399]$, $[0, 699] \times [0, 399] \times [0, 399]$ et $[0, 999] \times [0, 399] \times [0, 399]$.Voici ces sont les trois résultats :



(a) $n1 = 400$ (d) $n1 = 1000$

FIGURE 1 – Résultats d'accélération de la methode grain fin

On peut aussi comparér la vitesse d'accélération pour $threads = 4$

Taille de calcul	400	700	1000
Temps de calcul(s)	6.00	9.81	15.60

4 Version multithreads avec OpenMP (grain grossier)

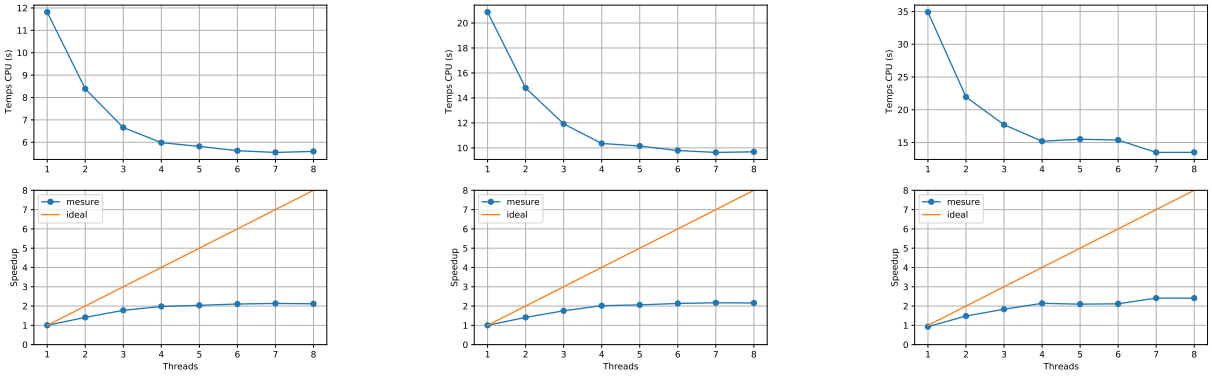
Pour la methode de grain grossier, j'ai aujouté un parallélisation autout la boucle en temps dans "main.cxx", et modifié la function de "Schema : :iteration()" et "Schema : :iteration_domaine" dans "scheme.cxx" pour que il puisse calculer en parallèle et transférer les résultats à état suivant.

Comme dans la partie de grain fin, j'ai lancé les calculs sur les 3 domaine différentes.Voici ces sont les trois résultats :

On peut aussi comparér la vitesse d'accélération pour $threads = 4$

Taille de calcul	400	700	1000
Temps de calcul(s)	5.98	10.35	15.2

Généralement, la méthode de grain grossier et la méthode de grain fin ont eu presque la meme performance. Juste pour la taille plus grande ($n1 = 1000$ dans cette situation), la éfficacité d'accélération de grain groissier(2.4 environ pour $threads = 7$ et $threads = 8$) est plus haute que la méthode de grain fin(2.2 environ pour $threads = 7$ et $threads = 8$).



(a) $n1 = 400$ (d) $n1 = 1000$

FIGURE 2 – Résultats d'accélération de la methode grain grossier