TP3

Laura Martin

1 Exemple OpenMP "grain fin"

Demonstration 1

(Pendant ce TP on fera toujours référence au temps réel, pas au temps user. Dans chaque cas on a répété plusieurs fois les calculs, et on montre seulement les valeurs moyens et/ou plus répétées. Les calculs de temps sont expresses en secondes.)
Pour le temps séquentiel on obtient en moyenne 3.20 sec. Si on parallélise le code avec n divisions, on attend un temps réel de $\frac{3.20}{n}$ sec. Donc pour $n = 3$ on attend 3.07 sec, mais expérimentalement on obtient 1.20 sec. Les 0.13 sec extras sont dues à la activation des autres $n-1$ cœurs de calcul.

2 Exemple OpenMP "grain fin (peut-être) amélioré"

Question 1

	Static	Dynamic
n/3	1.21	1.21
n/12	1.20	1.20
n/30	1.12	1.12
n/300	1.10	1.07

Fig. 1 – Temps du calcul avec la utilisation de *schedule*.

3 Exemple OpenMP "grain grossier"

Question 2

On avait deux possibilités pour mettre le *#pragma* avec les variables locaux ith, nth, etc. : La première était le mettre directement dans les fonctions *init* et *stat* pour paralléliser le code qu'ils aient dedans. Mais ce procédure implique réactiver des cœurs après les avoir désactive, donc on perd du temps. Alors la deuxième option est mettre *#pragma* dans le code principal et définir $i1$ et $i2$ dans chaque thread.

Comme *init* et *stat* demandent quels sont les itérations initial et final à faire, chaque thread fera les calculs relatifs à un morceau des vecteurs, en on n'allumera qu'une seule fois les cœurs, en gagnant du temps. Même avec ce statégie, la différence est très petit : 1.35 sec du sinus_fine vs. 1.34 sec du sinus_coarse_1.

Question 3

Pour montrer le temps de calcul pour chaque thread, on a modifié le code comme ci-dessus :

```
#pragma omp parallel
{
int ith=omp_get_thread_num();
int nth=omp_get_num_threads();
Timer T;
T.start();
int dn=n/nth;
int i1=ith*dn;
int i2=(ith+1)*dn;
if(ith==nth-1)
i2=n;

init(pos, v1, v2, i1, i2);

double m, e;

stat(v1, v2, i1, i2, m, e);
T.stop();
m = m/n;
e = sqrt(e/n - m*m);
std::cout << "m = " << m << " e = " << e << std::endl;
std::cout << "thead " << ith << "temps local: " << T.elapsed() << std::endl;
}
```

On peut pas sortir *std : :cout « "m = " « m « " e = " « e « std : :endl ;* du *#pragma* parce que *m* et *e* sont définies dans le *#pragma*. Mais ce n'est pas un problème parce que les erreurs locaux montrent aussi si le programme converge ou pas.

Le temps du sinus_fine a été le même que pour la question 2 (1.35 sec). Pour sinus_coarse_1 on voit que la division du travail n'a été très performante parce que le thread 0 prend moins temps que les autres pour faire son travail. Puisque le thread qui prend plus de temps utilise 1.34 sec pour finir, le temps global de la question 2 est 1.34 sec .

	thread 0	thread 1	thread 2
Temps local	$0.9089 sec$	$1.34261 sec$	$1.34161 sec$

Fig. 2 – Temps du calcul où on a décidé les valeurs des itérations initial et final.

4 Parallélisation du (mini-)code avec le modèle OpenMP "grain grossier"

Question 4

(Fichiers envoyés par mail)