TP2 - IF5-PAR Parallélisme et Architecture

Parallélisme intra-processeur – OpenMP

Les variables d'environnement qui permettent d'utiliser icc ne sont pas positionnées par défaut pour votre compte. Il faut donc exécuter les scripts suivant (et à ajouter dans votre dans votre .cshrc) :

1. pour icc : *source /nfs/user/pers/info/codesino/intel/iccvars.csh*
2. pour idb : *source /nfs/user/pers/info/codesino/intel/idbvars.csh*

Vous disposez alors de la commande icc et de son man.

Exercice 1 – Prise en main

Ecrire un programme C qui se *fork* en 4 threads openMP, chacun affichera son numéro de rang, puis à l'issue le programme affichera qu'il se termine.

Pour cela il faut :

- Définir la variable d'environnement OMP_NUM_THREAD : *setenv OMP_... 4*
- La directive *#include <omp.h>*,
- La directive *#pragma omp parallel* avant la région parallèle qui sera entre accolades,
- La fonction *int omp_get__thread_num ( )* pour obtenir le rang (numéro) d'un thread,
- Compiler sous *icc* avec l'option *–openmp*

Remarque : la fonction *int omp_get_num_threads( )* permet de connaître le nombre de thread déclaré par la variable d'environnement ou la dernière exécution de la fonction *omp_set_num_threads_nbr_threads( )*.

Exercice 2 – Variables privées

Modifiez (après l'avoir conservé sous un autre nom) l'exercice 1 afin que :

- chacun affiche le contenu d'une variable *VALEUR1* déclarée après le main et initialisée à 1000,
- chacun déclare privée affiche le contenu d'une variable *VALEUR2* déclarée après le main et initialisée à 2000. *VALEUR2* sera déclarée privée à l'entrée de chaque thread à l'aide de la directive *private (VALEUR2)* ajoutée à la fin de la ligne *#pragma omp parallel*. Chaque thread incrémentera la valeur de *VALEUR2,*
- Quelle est la valeur affichée par chaque thread ?
- Modifiez *private* par *firstprivate* et observer le résultat – Conclusion ?

Exercice 3 – Boucles parallèles

Modifiez (après l'avoir conservé sous un autre nom) l'exercice 1 (pas 2 !) :

- ajoutez une variable i après le main, elle servira d'indice de boucle,
- ajoutez for au *#pragma* de l'exercice 1 *: #pragma omp parallel for* (ATTENTION, il doit être placé juste devant la boucle),
- écrire une boucle *for* qui fait varier *i* de 1 à 20,
- chaque thread doit afficher la valeur de i en plus de son rang.

Exercice 3 – Calcul de PI

Transformez le calcul de π suivant afin de le rendre parallèle avec openMP (pour 2 threads). Attention au traitement des variable *som* et *x*.

Comparez les durées d'exécution entre la version mono-thread et la version 2 thread.
Que constatez-vous ? Votre processeur est-il dual-core ou seulement doté d'hyper-threading ?

Remarque : sous linux le fichier */proc/cpuinfo* permet d'obtenir des informations sur le(s) processeur(s).

Modifiez le code optimisé sachant que *#pragma parralel for reduction (+ :som) private (j)* permet de stocker dans *som* la somme des *som* privés de chaque thread, *j* est déclarée privée à chaque thread.

```
#include<stdio.h>
int main ()
{     static long nb_pas = 100000;
      double pas;
      int i; double x, pi, som = 0.0;
      pas = 1.0/(double) nb_pas;
      for (i=1;i<= nb_pas; i++){
            x = (i-0.5)*pas;
            som = som + 4.0/(1.0+x*x);
      }
      pi = step * som;
      printf("PI=%f\n",pi);
      return 0;
}
```

Pour mémoire :

```
long long readTSC () {
      long long t;
      asm volatile (".byte 0x0f,0x31" : "=A" (t)); return t; }
double dtime() { return (double) readTSC(); }
```