Introduction à la programmation parallèle avec openMP

Gaëtan Hello

Université d'Evry Val d'Essonne
UFR S&T - Laboratoire de Mécanique et d'Energétique d'Evry
gaetan.hello@ufrst.univ-evry.fr

M2 GM CS - CS93 - 2013

1. Principes
   - Programmation parallèle
   - Parallélisation avec openMP

Programmation séquentielle/parallèle

En programmation séquencielle, le programme informatique est conçu de sorte à ce que les opérations à réaliser se déroulent les unes à la suite des autres. Le programme s'exécute alors sur un unique *thread* (fil d'exécution) supporté par une seule unité de calcul.

La programmation parallèle consiste elle à concevoir des programmes informatiques où plusieurs opérations pouront être réalisées simultanément. L'exécution de tels programmes repose sur la possibilité de décomposer les traitements sur plusieurs threads gérés par des unités de calcul distinctes.

Qu'est ce qu'openMP ?

openMP :

- est le raccourci de "open Multi Processing",
- est l'API (Application Protocol Interface) de-facto pour la création de programmes parallèles sur machines à mémoire partagée,
- est portable : multi-OS (linux, windows, macOS, ...), multi-architecture (nombreux types de processeurs), multi-langage (C/C++, fortran, ...) et supportés par de nombreux compilateurs (par exemple pour le C/C++ : gcc, MS VC++, icc, ...),
- comprend des directives de compilation, des librairies et des variables d'environnement,
- n'est pas une simple librairie, le compilateur (C/C++, fortran,...) utilisé doit avoir été adapté pour pouvoir gérer les directives.

Qu'est ce qu'openMP ?

Il généralement pertinent d'utiliser openMP pour réaliser de la programmation parallèle lorsque l'on :

- exploite une machine à mémoire partagée (typiquement un ordinateur à processeur(s) multicoeurs. Exemple : PC quadri-processeurs intel xeon E7-8890 v2 à 15 coeurs + technologie d'hyperthreading $\rightarrow$ 4*15*2=120 threads disponibles),
- on souhaite modifier un algorithme séquentiel pour le rendre parallèle en appportant peu de modifications au code initial,

openMP seule "solution" ?

En plus de l'approche proposée par openMP, on peut distinguer 3 autres principales manières d'envisager la programmation parallèle :

- *hand threading* : le développeur peut et doit gérer lui-même la répartition des opérations sur différents threads :
  ☺ contrôle important,
  ☹ plus complexe à gérer qu'openMP,
- MPI (*Message Passing Interface*) : le développeur gère la communication entre les différentes unités de calcul (mémoire partagée ou distribuée) :
  ☺ *scalability* (facilité de monter en charge en ajoutant des ressources matérielles supplémentaires) pour des architectures à mémoire distribuée,
  ☹ plus complexe à gérer qu'openMP, performance moindre en mémoire partagée.

openMP seule "solution" ?

En plus de l'approche proposée par openMP, on peut distinguer 3 autres principales manières d'envisager la programmation parallèle :

- GPGPU (*General Purpose processing on Graphics Processing Units*) : l'architecture nativement parallèle des cartes graphiques est exploitée pour y exécuter des calculs autres que ceux liés au seul traitement de l'affichage. Nvidia propose la plateforme CUDA (Compute Unified Data Architecture) pour exploiter ses cartes graphiques (ex : carte nvidia tesla K20 à 2496 coeurs) :
  ☺ performances potentiellement beaucoup plus importantes que sur CPU pour les algorithmes massivement parallélisables (jusqu'à $\approx$ 20 fois),
  ☹ plus complexe à gérer qu'openMP, pas encore de standard.

Parallélisation avec openMP

- une application openMP débute sur un premier thread (master thread),
- lors de l'exécution du programme, l'application peut rencontrer des régions parallèles où le master thread crée des thread teams,
- le master thread continue en parallèle,
- à la fin de la région parallèle, les thread teams cessent leurs traitements et le master thread est le seul à continuer,
- il est possible de créer des nested thread teams dans chaque chaque thread teams issues du master thread.

Parallélisation avec openMP

- la programmation de la parallélisation avec openMP est algorithmiquement et syntaxiquement simple,
- le développeur utilise simplement des pragmas et des routines du runtime openMP dans son programme (en C/C++ ici),
- les pragmas sont considérées par le compilateur si celui-ci est compatible avec openMP (il les ignore sinon),
- les routines runtime réclament elles d'être incluses dans le programme (#include <omp.h>).

Parallélisation avec openMP

- les pragmas sont des directives adressées au compilateur pour qu'il sache que le code possède des attributs parallèles dans une portion donnée du code,
  ex : #pragma omp parallel for
- les routines runtime sont des méthodes permettant de : fixer des paramètres de la parallélisation, récupérer des informations sur l'état de la parallélisation, ou réaliser des opérations de synchronisation,
  ex : omp_set_num_threads(10) ;

2 Quelques fonctionalités d'openMP

- runtime routines
- pragma
- environment variables

runtime routines

```
#include <omp.h>
...
int n=omp_get_num_threads();
int n=omp_get_max_threads();
int n=omp_get_num_procs();
int n=omp_get_thread_num();
omp_set_num_threads(n);
```

pragma

```
#include <iostream>

void main()
{
   #pragma omp parallel
   {
      std::cout<<"le calcul parallele c'est super !\n";
      system("pause");//pour windows !!!
   }
}
```

pragma

```
#include <iostream>

void main()
{
   #pragma omp parallel
   std::cout<<"le calcul parallele c'est super !\n";
   system("pause");//pour windows !!!
}
```

pragma

```
#include <iostream>

void main()
{
   int n=1E7;
   double* tab=new double[n];
   #pragma omp parallel
   {
      for(int i=0;i<n;i++)
      {
         tab[i]=cos(double(i));
      }
      std::cout<<"fini !\n";
   }
   system("pause");//pour windows !!!
}
```

pragma

```
#include <iostream>

void main()
{
   int n=1E7;
   double* tab=new double[n];
   #pragma omp parallel
   {
      #pragma omp for
      for(int i=0;i<n;i++)
      {
         tab[i]=cos(double(i));
      }
   }
   std::cout<<"fini !\n";
   system("pause");//pour windows !!!
}
```

pragma

```
#include <iostream>

void main()
{
   int n=1E7;
   double* tab=new double[n];
   #pragma omp parallel for
   for(int i=0;i<n;i++)
   {
      tab[i]=cos(double(i));
   }
   std::cout<<"fini !\n";
   system("pause");//pour windows !!!
}
```

pragma

```
#pragma omp parallel
{
   #pragma omp sections
   {
      #pragma omp section
      for(int i=0;i<n;i++)
      {
         tab1[i]=cos(double(i));
      }
      #pragma omp section
      for(int i=0;i<n;i++)
      {
         tab2[i]=sin(double(i));
      }
   }
}
```

environment variables

Variables visibles par l'OS

```
OMP_NUM_THREAD
OMP_THREAD_LIMIT
OMP_MAX_ACTIVE_LEVELS
OMP_WAIT_POLICY
OMP_NESTED
OMP_DYNAMIC
```

3 Exercices