

Manip. AMINA Workshop 19-11-2010

Prof. M. AKIL & R. MAHMOUDI

Introduction

Ce tutorial vous présentera la programmation parallèle faite avec OpenMP. Vous trouverez dans ce manuscrit une présentation générale des concepts de base dont vous avez besoin pour intégrer OpenMP dans vos programmes, ainsi qu'un simple exemple commenté d'utilisation d'OpenMP. Pour les mots, concepts qui ne sont pas définis directement ici, une définition plus complète existe sur ces liens : www.openmpwiki.fr.nf / www.openmp.fr.nf.

Vous pouvez également télécharger la présentation théorique ainsi que le code source de l'application depuis www.mramzi.net.

Qu'est ce que OpenMP

OpenMP : Open Multi-Processing est un ensemble de modules permettant la réalisation d'applications utilisant le multi-threading sur C/C++ ou Fortran et il est supporté par plusieurs plateformes : Unix et Windows

Pourquoi utiliser OpenMP

Avec l'émergence des processeurs multicores, les performances des ordinateurs se sont nettement améliorées, il s'agit donc de mettre à profit cette puissance de calcul au niveau algorithmique. OpenMP va justement nous permettre d'exploiter ces performances grâce à une gestion contrôlée des différents cœurs d'un seul processeur ou de plusieurs processeurs. En effet cette API de programmation parallèle va nous donner une certaine transparence dans l'usage de ces architectures avec plus d'efficacité.

Histoire d'OpenMP

OpenMP Version 3.0, Mai 2008.

OpenMP Version 2.5 a combiné C/C++/Fortran, 2005.

OpenMP Version 2.0 en pour Fortran et C/C++, 2000 et 2002.

OpenMP pour C/C++, Octobre 1998.

OpenMP 1.0 pour Fortran, Octobre 1997.



Compilateur vs OpenMP

Cons./Source	Compilateur	Information
GNU	gcc (4.3.2)	Open source - Linux, Solaris, AIX, MacOSX, Windows Compiler avec : -fopenmp (http://gcc.gnu.org/projects/gomp/)
IBM	XL C/C++ / Fortran	AIX and Linux. (http://www-01.ibm.com)
Oracle	C/C++ / Fortran	Oracle Solaris Studio compilers and tools pour Solaris et Linux. Compiler avec -xopenmp (http://www.oracle.com/)
Intel	C/C++ / Fortran (10.1)	Windows, Linux, et MacOSX. Compiler avec -Qopenmp sous Windows, Ou bien -openmp sous Linux / Mac OSX (http://software.intel.com/en-us/articles/intel-compilers/)
Portland Group Compilers	C/C++ / Fortran	PGI Compilers Compiler avec -mp (http://www.pgroup.com/resources/openmp.htm)
Absoft Pro Fortran	Fortran	Fortran 95 compiler (Version 11.1) pour Linux, Windows and Mac OS X Compiler avec -openmp. (http://www.absoft.com/SMP_Solutions.html)
Lahey/Fujitsu Fortran 95	C/C++ / Fortran	(http://www.compunity.org/resources/compilers)
PathScale	C/C++ / Fortran	PathScale Compiler pour Linux 32/64 bits (http://www.pathscale.com/ws/docs/3.1/UserGuide.pdf)
HP	C/C++ / Fortran	(http://docs.hp.com/en/B3909-90031/index.html)
MS	Visual Studio 2008 C++	(http://msdn.microsoft.com/en-us/library/tt15eb9t.aspx)
Cray	Cray C/C++ and Fortran	Cray XT series Linux environment. (http://docs.cray.com/browsehome.html)

Comment intégrer OpenMP dans votre environnement de travail

Comment ajouter OpenMP à votre environnement de travail Linux
La version 4.2, GCC supporte OpenMP. On l'installe donc simplement avec le « distribution's package manager ». La compilation de programmes utilisant OpenMP peut par exemple se faire avec la commande :

```
$ gcc -fopenmp -o hello hello-openmp.c
```

Maintenant si vous voulez absolument travailler sous windows avec un simple éditeur de texte comme blocnote ou notepad++ et le compilateur gcc.

Pour commencer direction le site web **www.mingw.org**
Aller dans la section downloads (en bas, à gauche de la page).
Aller dans la section « **Automated MinGW Installer/mingw-get-inst** »
Double click sur **mingw-get-inst20101030**
Lancer le fichier « **mingw-get-inst20101030.exe** » après l'avoir sauvegardé
Si l'installation se termine avec succès, vous pouvez mettre à jours l'ensemble des packages : Depuis une invite de commande, accéder à « **C:\MinGW\bin** »
Lancer « **mingw-get update** »

Quant à la partie configuration (sous windows xp):

Clique droit sur poste de travail et propriété.
Paramètre système avancé
variables d'environnements
variables système
Chercher "**PATH**" et modifier.
Aller à la fin de la ligne et rajouter "**;c:\mingw\bin**" sans les guillemets.

Sous Windows Vista/Seven la procédure est pratiquement la même :

Clique droit ordinateur et propriété
Paramètre système avancé
variables d'environnement
variables système
Chercher "**PATH**" et modifier
Aller à la fin de la ligne et rajouter "**;c:\mingw\bin**" sans les guillemets.

Tester votre configuration

Nous allons créer le programme suivant avec un simple éditeur de texte :

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define _OMP_NUM_THREADS 2

int main ()
{
printf (" \n AMINA WOKSHOP 2010 - HELLO WORD with OpenMP \n \n");

#pragma omp parallel sections
{
#pragma omp section
{ printf (" \n Hello World 1 \n \n"); }

#pragma omp section
{ printf(" \n Hello World 2 \n \n"); }

#pragma omp section
{ printf(" \n Hello World 3 \n \n"); }

}
}
```

Enregistrer votre fichier (avec l'extension .c) dans un répertoire 'AMINA' sous 'C:\'. Pour tester votre code :

Ouvrez une fenêtre MSDOS (cmd.exe)

Placer vous dans le répertoire 'AMINA'

Compiler votre code: **gcc -openmp hello.c -o hello.exe**

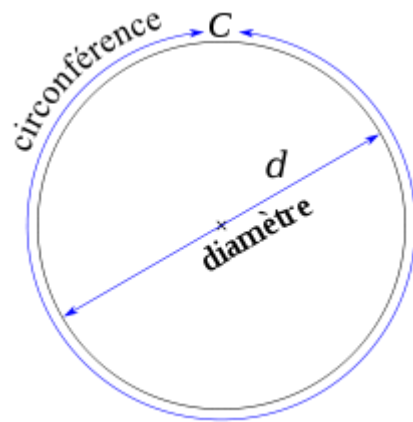
Enfin pour l'exécuter taper directement : **hello.exe**

Vous devez avoir un résultat de ce type sur vos écrans :

```
C:\AMINA\Lab1 HELLO>gcc -openmp hello.c -o hello.exe
C:\AMINA\Lab1 HELLO>hello.exe
----- AMINA WOKSHOP 2010 - HELLO WORD with OpenMP -----
Hello world 1
Hello world 2
Hello world 3
```

Calculer Pi

Pi (ou parfois **constante d'Archimède**) est un nombre, généralement représenté par la lettre grecque du même nom π , et généralement défini comme étant le rapport entre la circonférence d'un cercle quelconque et son diamètre, en géométrie euclidienne. On peut également le définir comme le rapport entre la superficie d'un cercle et le carré de son rayon. La valeur de pi arrondie à 10^{-6} est 3,141593 en écriture décimale. Des nombreuses formules, dans des domaines tels que la physique, l'ingénierie et bien sûr les mathématiques, impliquent π , qui est une des constantes les plus importantes des mathématiques.



π est un nombre irrationnel, c'est-à-dire qu'on ne peut pas l'exprimer comme un rapport de deux nombres entiers ; ceci entraîne que son écriture décimale n'est ni finie, ni périodique. C'est aussi un nombre transcendant, ce qui signifie qu'il n'existe pas de polynôme non nul à coefficients entiers dont π soit une racine ; la preuve de ce résultat en 1882 est due à Ferdinand von Lindemann. La détermination d'une valeur approchée suffisamment précise de π et la compréhension de sa nature sont des enjeux qui ont traversé l'histoire des mathématiques ; la fascination exprimée par certains envers ce nombre l'a même fait entrer dans la culture populaire.

Pour calculer Pi, nous proposons d'utiliser:

1. la méthode d'intégration numérique

```
// Pi Integration numérique
double PiIntegratiOnNumerique()
{
    long i;
    double x, sum = 0.0;
    step = 1.0/(double) ITERATIONS;
    //le travail commence ici
    for (i=0; i< ITERATIONS; i++){
        x = (i+0.5)*step;
        sum = sum + 4.0/(1.0 + x*x);
    }
    pi = step * sum;
    // le travail se termine ici
    return pi;
}
```

2. la méthode de James Gregory

```
// Pi JamesGregory method
double PiJamesGregory()
{
    // Pi par formule James Gregory
    // Pi = 4( 1 - 1/3 + 1/5 - 1/7 + 1/9 ...)

    pi = 0.0;
    int doAdd = 1;
    int i ;

    // le travail commence ici
    for ( i = 1; i < ITERATIONS; i+=2)
    {

        if ( doAdd == 1 ) {
            pi += 1.0/i;
            doAdd = 0;}
        else {
            pi -= 1.0/i;
            doAdd = 1;
        }
    }
    // le travail se termine ici

    return pi;
}
```

Pour profiler le temps, nous ferons appel à la fonction clock de la « time.h ».

Compiler puis exécuter le code suivant après l'avoir compléter par les deux fonctions : PiIntegrati0nNumerique() et PiJamesGregory() introduites précédemment.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#define ITERATIONS 100000000

double step, pi;

float rx[ITERATIONS];
float ry[ITERATIONS];

.....
.....
.....

int main ()
{
printf (" \n ----- AMINA WOKSHOP 2010 - Pi Integration ----- \n \n");

clock_t tempsDebut;
clock_t tempsFin;
clock_t tempsTotalDebut = clock();
clock_t tempsTotalFin;

printf (" \n PiIntegrati0nNumerique \n");
tempsDebut = clock( );
pi = PiIntegrati0nNumerique();
tempsFin = clock( );
printf(" \n Pi integration numerique = %f\n",pi);
printf("%s% 10ld\n", " \n temps necessaire (en ms) = ",tempsFin - tempsDebut);
printf("\n FIN PiIntegrati0nNumerique \n");

printf(" \n DEBUT PiJamesGregory \n \n");
tempsDebut = clock( );
pi = PiJamesGregory();
tempsFin = clock( );

printf(" Pi James Gregory = %f\n",pi*4.0);
printf("%s% 10ld\n", " \n temps necessaire (en ms) = ",tempsFin - tempsDebut);
printf("\n FIN PiJamesGregory \n");

tempsTotalFin = clock();
printf("\n %s% 10ld\n", "temps Total ***** = ",tempsTotalFin - tempsTotalDebut);
}
```

Insérer la fonction PiIntegrati0nNumerique()
Insérer la fonction double PiJamesGregory()

Après l'exécution, vous devez avoir un résultat de ce type sur vos écrans :

```
C:\AMINA\Lab2 Pi>a.exe
----- AMINA WOKSHOP 2010 - Pi Integration -----
PiIntegrati0nNumerique
Pi integration numerique = 3.141593
temps necessaire (en ms) =      1903
FIN PiIntegrati0nNumerique
DEBUT PiJamesGregory
Pi James Gregory = 3.141593
temps necessaire (en ms) =      952
FIN PiJamesGregory
temps Total ***** =      2886
```

Donner le temps total d'exécution que vous avez obtenue en précisant le type de votre machine.

Injecter du parallélisme avec OpenMP

Pour tester le principe des sections parallèle avec OpenMP, modifier la fonction main en ajoutant la primitive « pragma section » de la manière suivante :

```
int main ()
{
printf (" \n - AMINA WOKSHOP 2010 - Pi Integration with OpenMP - \n \n");

clock_t tempsDebut;
clock_t tempsFin;
clock_t tempsTotalDebut = clock();
clock_t tempsTotalFin;

#pragma omp parallel sections ← Ajouter le pragma parallel section
{
#pragma omp section
{ printf (" \n DEBUT SECTION 1 \n");
tempsDebut = clock( );
pi = PiIntegrati0nNumerique();
tempsFin = clock( );
printf(" \n Pi integration numerique = %f\n",pi);
printf("%s% 10ld\n", " \n temps necessaire (en ms) = ",tempsFin - tempsDebut);
printf("\n FIN SECTION 1 \n");
}
}
```

```

#pragma omp section
{ printf(" \n DEBUT SECTION 2 \n \n");
  tempsDebut = clock();
  pi = PiJamesGregory();
  tempsFin = clock();

  printf(" Pi James Gregory = %f\n",pi*4.0);
  printf("%s%10ld\n", " \n temps necessaire (en ms) = ",tempsFin - tempsDebut);
  printf("\n FIN SECTION 2 \n");
}
}
tempsTotalFin = clock();
printf("\n %s%10ld\n", "temps Total ***** = ",tempsTotalFin - tempsTotalDebut);
}

```

Après l'exécution de votre code : vous devez avoir un résultat de ce type :

```

C:\AMINA\Lab3 PiOmpSec>mp.exe
----- AMINA WOKSHOP 2010 - Pi Integration with OpenMP -----
DEBUT SECTION 1
Pi integration numerique = 3.141593
temps necessaire (en ms) =      1887
FIN SECTION 1
DEBUT SECTION 2
Pi James Gregory = 3.141593
temps necessaire (en ms) =      967
FIN SECTION 2
temps Total ***** =      2854

```

Donner le temps total d'exécution que vous avez obtenue.
 Augmenter le nombre d'itération puis tester de nouveau votre code. Que remarquez-vous ?

Maintenant nous désirons tester le module « parallel for » avec OpenMP, pour cela modifier la fonction `PiIntegratiOnNumerique()` en ajoutant le « pragma parallel for » de la manière suivante :

```
// Pi Integration numérique
double PiIntegratiOnNumerique()
{
    long i;
    double x, sum = 0.0;
    step = 1.0/(double) ITERATIONS;
    //le travail commence ici
    #pragma omp parallel for
    for (i=0; i< ITERATIONS; i++){
        x = (i+0.5)*step;
        sum = sum + 4.0/(1.0 + x*x);
    }
    pi = step * sum;
    // le travail se termine ici
    return pi;
}
```

Ajouter le pragma parallel for

Apporter les même modifications pour la fonction `PiJamesGregory()` :

```
double PiJamesGregory()
{
    // Pi par formule James Gregory
    // Pi = 4( 1 - 1/3 + 1/5 - 1/7 + 1/9 ...)

    pi = 0.0;
    int doAdd = 1;
    int i ;

    // le travail commence ici
    #pragma omp parallel for
    for ( i = 1; i < ITERATIONS; i+=2)
    {

        if ( doAdd == 1 ) {
            pi += 1.0/i;
            doAdd = 0;}
        else {
            pi -= 1.0/i;
            doAdd = 1;
        }
    }
    // le travail se termine ici

    return pi;
}
```

Après l'exécution de votre code, vous devez retrouver un résultat de ce type :

```
DEBUT PiIntegrati0nNumerique
Pi integration numerique = 3.141593
temps necessaire (en ms) =      1872
FIN PiIntegrati0nNumerique
DEBUT PiJamesGregory
Pi James Gregory = 3.141593
temps necessaire (en ms) =      952
FIN PiJamesGregory
temps Total ***** =      2839
```

Donner le temps total d'exécution que vous avez obtenue.

Augmenter le nombre d'itération puis tester de nouveau votre code. Que remarquez-vous ?

Enfin essayer de combiner les deux primitives (parallel section et parallel for) et de refaire les testes ! Que remarquez-vous ?