INF7235 : Laboratoire #4
Multiplication parallèle de polynomes

13 (ou 20) février 2017

La figure **??** présente les spécifications d'un type abstrait `Polynomes` pour la manipulation de **polynomes de taille arbitraire**.

Pour obtenir les fichiers qui vous sont fournis — **à compiler et exécuter sur `japet`!** :

```
$ git clone http://www.labunix.uqam.ca/~tremblay/git/PolynomesTBB.git
```

Ce que vous devez faire

1. Le type abstrait `Polynomes` exporte diverses opérations. Plusieurs de ces opérations sont déjà mises en oeuvre, tant de façon séquentielle que parallèle — dans ce dernier cas à l'aide des constructions `parallel_for` et `parallel_reduce` des *Threading Building Blocks* d'Intel.

   Votre tâche est de compléter la mise en oeuvre, tant séquentielle que parallèle, **du produit de polynomes — opération `fois`**.

   Note : L'opération `selectionnerMode` permet de sélectionner le mode d'exécution : `SEQUENTIEL` ou `PARALLELE`. Ce mode peut être changé durant l'exécution.

2. Lorsque vos solutions séquentiel et parallèle pour `fois` exécuteront tous les tests avec succès, exécutez le programme de mesures des performances (`mesurer-polynomes.c` : voir plus bas) pour déterminer **à partir de quelle valeur de `N` vous réussissez à obtenir une accélération**, et pour quel nombre de *threads*.

```
/** Type abstrait simple pour des polynomes
    de taille arbitraire a
    coefficients reels (double).

    Note: Version a la C, donc sans classe,
    mais a paralleliser avec
    TBB/C++.

    @name Polynomes
    @author Guy Tremblay
*/

#ifndef POLYNOMES_H
#define POLYNOMES_H

typedef struct {
  int degre;
  double* coefficients;
} Polynome;


Polynome polynome ( int degre, ... );
Polynome polynome_( int degre, double coefficients[] );

Polynome plus( Polynome p1, Polynome p2 );
Polynome fois( Polynome p1, Polynome p2 );

bool egaux( Polynome p1, Polynome p2 );

void dump( char* msg, Polynome p );


// Pour selectionner le mode d'execution.
enum Mode { SEQUENTIEL, PARALLELE };
void selectionnerMode( Mode m );

#endif
```

Figure 1: Interface des opérations pour le type abstrait `Polynome`.

Ce qui vous est fourni

Un certain nombre de fichiers vous sont fournis, fichiers qui doivent être compilés et exécutés **sur la machine** `japet` :

1. `polynomes.h` : le fichier d'interface présenté à la figure **??**.
2. `polynomes.c` : le corps du module `Polynomes`, où les opérations spécifiées dans la partie interface (`polynomes.h`) sont mises en oeuvre, sauf pour l'opération `fois`.
3. Un (petit) programme de tests, `tester-polynomes.c`, défini à l'aide du cadre de tests `MiniCUnit`.
4. `MiniCUnit.h` et `MiniCUnit.c` : le code pour le cadre de tests.
5. `mesurer-polynomes.c` : un fichier contenant un programme pour mesurer le temps d'exécution de la multiplication, qui compare la version séquentielle et la version parallèle.
6. `makefile` : un fichier qui permet d'automatiser la compilation et l'exécution des fichiers et programmes.

   Les principales commandes associées à l'utilisation de ce `makefile` sont les suivantes (commandes exécutées au niveau du *shell* Unix) :

   - «`make compile`» : compile *l'ensemble des fichiers et programmes.*

     Note : `compile` est la cible par défaut du `makefile`, donc «`make`». sans argument équivaut à «`make compile`».
   - «`make tests-s`» ou «`make tests-p`» ou «`make tests`» : compile les fichiers et exécute les tests pour la version séquentielle, ou la version parallèle, ou pour les deux versions.
   - «`make mesures N=k`» : compile les fichiers pour le programme `mesurer-polynomes.c` puis l'exécute pour un polynome de taille «`k`».
   - «`make clean`» : fait le ménage, c'est-à-dire, supprime les fichiers qui peuvent être générés de façon automatique.

<