INF5171: Cours–Laboratoire #10
Opérations parallèles sur des polynomes en TBB/C++

28 et 30 novembre 2017
13h30–16h30
PK-S1560/PK-S1565

Le but de ce laboratoire est de vous familiariser avec l'utilisation des *Threading Building Blocks* en C++.

La figure 1 présente les spécifications d'un type abstrait `Polynomes` pour la manipulation de **polynomes de taille arbitraire** — voir le Chapitre 9 des notes de cours «Exemples illustrant l'approche PCAM», section «9.3 Opérations sur des polynomes».

Divers fichiers vous sont fournis sous forme d'un dépôt `git`, que vous devez obtenir en exécutant la commande suivante **sur** `japet` :

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

Ce que vous devez faire

a. Le type abstrait `Polynomes` exporte diverses opérations que vous devez mettre en oeuvre **en parallèle**, et ce à l'aide de diverses constructions des *Threading Building Blocks* d'Intel — `parallel_for`, `parallel_reduce`, `task_group`, etc.

   Plus précisément, dans le fichier `polynomes.c` qui vous est fourni, les mises en oeuvre séquentielles sont fournies. Il vous faut donc compléter les parties parallèles, qui sont indiquées par un commentaire «`// A COMPLETER!`» ou «`// A MODIFIER...`».

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

   Voici l'ordre suggéré, ainsi qu'une contrainte à respecter pour `egaux` :

   (a) `plus`

   (b) `fois`

   (c) `eval`

   (d) `egaux` : Pour cette fonction, vous devez utiliser du parallélisme **à granularité fine** — donc une tâche pour chaque élément — et ce en utilisant un `task_group` et les fonctions `run` et `cancel`. Cette dernière fonction pourra vous permettre de terminer le traitement des différentes tâches **aussitôt que possible**, donc de façon **court-circuitée** — en d'autres mots, la fonction retourne `false` dès qu'on a identifié deux coefficients qui diffèrent!

b. Lorsque votre solution parallèle exécutera 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 ce pour quel nombre de *threads*.

c. Dans les versions récentes de TBB, c'est le `auto_partitioner()` qui est utilisé par défaut pour `parallel_for` et `parallel_reduce`, donc ces appels sont équivalents :

```
parallel_for( r, lambdaExpr );
parallel_for( r, lambdaExpr, auto_partitioner() );
```

   Par contre, dans les anciennes versions de TBB, c'était le `simple_partitioner()` qui était le défaut. On peut obtenir l'ancien comportement en modifiant le `makefile` :

```
# Avant modification
.c.o:
        $(CC) -c $(CFLAGS) $<
# Apres modification
.c.o:
        $(CC) -D TBB_DEPRECATED=1 -c $(CFLAGS) $<
```

   Faites cette modification et exécutez à nouveau le programme de mesures des performances. Qu'en concluez-vous? Pourquoi?

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

    Note: Il s'agit d'une version "a la C", donc sans classe, 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 );

double eval( Polynome p, double x )

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` :

a. `polynomes.h` : le fichier d'interface présenté à la figure 1.

b. `polynomes.c` : le corps du module `Polynomes`, où les opérations spécifiées dans la partie interface (`polynomes.h`) sont définies, mais uniquement de façon séquentielle.

c. Un (petit) programme de tests, `tester-polynomes.c`, défini à l'aide du cadre de tests `MiniCUnit`.

d. `MiniCUnit.h` et `MiniCUnit.c` : le code pour le cadre de tests.

e. `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.

f. `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-plus`», «`make tests-fois`» ou «`make tests-egaux`» : compile les fichiers et exécute les tests pour la version parallèle de la fonction `plus`, `fois` ou `egaux`.
   - «`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.