UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Máster Universitario en Computación en la Nube y de Altas Prestaciones

Tecnología de la Programación Paralela (TPP).

Ejercicios Threading Building Blocks Sesión 2

Introducción

Esta parte, continuación de la Sesión 1, contiene una serie de ejercicios propuestos de Threading Building Blocks. Adjunto a este enunciado y para la mayoría de ejercicios existe un código fuente base de la versión secuencial que se ha de paralelizar.

La librería TBB con la que vamos a trabajar es la que viene con la distribución de Intel, (`oneTBB`). Por ello, es posible que antes de comenzar haya que leer las variables de entorno para trabajar con TBB:

```
source /opt/intel/oneapi/tbb/latest/env/vars.sh
```

aunque también es posible que no sea necesario porque ya están inicializadas dichas variables.

A partir de aquí se puede trabajar con TBB, tanto con el compilador de Intel como con el de GNU. Un programa escrito en TBB se puede compilar con el compilador de GNU así:

```
g++ -o programa programa.cpp -ltbb
```

o con el propio compilador de Intel:

```
icpc -o programa programa.cpp -qtbb
```

Obsérvese que para Intel es una opción del compilador mientras que para GNU es una librería. En principio, nosotros trabajaremos con el compilador de GNU, `g++`.

Ejercicio 1

Paralelizar con TBB el siguiente código:

```
double SerialSumFoo( double a[], size_t n ) {
  double sum = 0;
  for( size_t i=0; i!=n; ++i )
    sum += Foo(a[i]);
  return sum;
}
```

donde la función `Foo` puede ser la siguiente:

```
double Foo( double f ) {
  unsigned int u = (unsigned int) f;
  double a = (double) rand_r(&u) / RAND_MAX;
  a = a*a;
  return a;
}
```

Existe un código, `ejercicio1.cpp`, que se puede utilizar para implementar este ejercicio.
Este ejercicio hay que resolverlo de dos maneras:

1. utilizando la forma *imperativa* (creando la clase *body*); y
2. utilizando la versión *funcional* (con expresiones *lambda*).

Ejercicio 2

Paralelizar con TBB el siguiente código:

```
long SerialMinIndexFoo( const double a[], size_t n ) {
  double value_of_min = DBL_MAX;
  long index_of_min = -1;
  for( size_t i=0; i<n; ++i ) {
    double value = Foo(a[i]);
    if( value < value_of_min ) {
      value_of_min = value;
      index_of_min = i;
    }
  }
  return index_of_min;
}
```

La implementación que aparece en las transparencias está expresada en forma imperativa. En este ejercicio hay que implementar la forma **funcional**. Existe un código, `ejercicio2.cpp`, que se puede utilizar para implementar este ejercicio.

Ejercicio 3

Paralelizar con TBB un código que, dada una secuencia de números reales, devuelva otra secuencia con los cuadrados de dichos números. El código del fichero `ejercicio3.cpp` contiene una versión secuencial que genera dicha secuencia.

Hay que observar que el bucle a paralelizar en la función `SerialApplyFooList` posee un número indeterminado de iteraciones, tantas como elementos en la lista, y el número de elementos no se conoce a priori. Por lo tanto, no sirven los algoritmos vistos hasta ahora, es decir, `parallel_for`. El algoritmo a utilizar aquí es `parallel_for_each`, que sería la versión paralela del algoritmo de la STL de C++ `for_each`. Para utilizar este algoritmo es necesario que exista un *iterador* (`Iterator`) sobre la lista de elementos. Así pues, en la nueva función a construir (`ParallelApplyFooList`) tendremos que crear un contenedor para poder disponer de un *iterador* que se pueda utilizar como primer argumento de la función `parallel_for_each`. Se puede utilizar, por ejemplo, el contenedor `list` de C++.

Ejercicio 4

Paralelizar con TBB el código facilitado en el fichero `actividad4.cpp` teniendo en cuenta que se trata de un *pipeline*. Para ello, debéis copiar el código que resuelve este problema y que tenéis en las transparencias de clase. La parte del código marcada como "Resolucion secuencial" es la que debería ser sustituida por el código paralelo.

Las variables que faltan en el fichero son las siguientes:

```
size_t next_buffer = 0;
char last_char_of_previous_buffer = ' ';
static const size_t n_buffer = 4;
MyBuffer buffer[n_buffer];
```

En la carpeta existe un fichero (`fichero_entrada`) con un texto con el que trabajar.