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 1

Introducción

Esta parte contiene una serie de ejercicios propuestos de Threading Building Blocks. Adjunto a este enunciado y para la mayoría de actividades 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 utilizando el algoritmo `parallel_for`:

```
void SerialApplyFoo( double a[], size_t n ) {
  for( size_t i=0; i<n; ++i )
    Foo(a[i]);
}
```

El código para la función `Foo` puede ser:

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

Utilizad el fichero base facilitado en la carpeta `material_tbbs1` y las transparencias de clase.

Se pide dos versiones de este programa, la primera sin expresiones *lambda* y la segunda con expresión *lambda*.

Ejercicio 2

Paralelizar con TBB el programa facilitado en el código `ejercicio2.cpp` de la carpeta `material_tbbs1`. Se recomienda seguir estos dos pasos:

1. El código paralelo consistirá en una llamada al algoritmo `parallel_for`, tal como se hizo en el ejercicio anterior, donde el primer argumento es de tipo `blocked_range` y el segundo es un objeto función. Este segundo argumento puede ser la llamada al constructor de una clase que implementaremos en el siguiente punto.

2. Construir una clase con el operador de acceso a función `operator()` implementado. Este operador recibe como argumento el `blocked_range` sobre el que va a trabajar. La implementación consiste en el bucle para la `v` que se quiere paralelizar. Cada uno de los objetos (`tam`, `M`, `media` y `desvt`) tendrán que ser miembros de la clase y ser inicializados mediante un constructor.

Esta actividad, al igual que la anterior, se compone de dos tareas. La primera consiste en la implementación siguiendo los pasos anteriores, es decir, sin la utilización de expresiones *lambda*. Para la segunda tarea se utilizarán expresiones *lambda*.

Ejercicio 3

Dado el algoritmo de Cholesky facilitado en la carpeta `material_tbbs1`, compiladlo mediante:

```
g++ -o cholesky cholesky.cpp ctimer.c -lblas -llapack
```

y probadlo.

Tarea 1: Copiad el algoritmo `cholesky.cpp` en uno nuevo, `cholesky_tarea1.cpp`, y paralelizadlo de la siguiente manera:

1. Paralelizad los dos bucles intermedios (para `i`) mediante el algoritmo `parallel_for` y creando las clases *body* pertinentes.
2. Sobre la paralelización anterior, paralelizar el bucle interno (para `j`) al segundo bucle para `i` para que se ejecuten las multiplicaciones en paralelo.

Tarea 2: Copiad de nuevo el algoritmo `cholesky.cpp` en un fichero llamado `cholesky_tarea2.cpp`, y paralelizadlo tal como se ha hecho en la tarea anterior pero utilizando en este caso expresiones Lambda.

Tarea 3: Copiad el código original `cholesky.cpp` en uno nuevo, `cholesky_tarea3.cpp`, y paralelizadlo utilizando la clase `blocked_range2d` para el segundo bucle intermedio. En este caso se deben utilizar directamente las expresiones Lambda para paralelizarlo.