Programación Concurrente
Práctico Taller 1: Region Labeling Problem

J. Blanco, N. Wolovick

26 de mayo de 2003

El problema 3.26 de [1] se reproduce a continuación:

> El siguiente problema de etiquetado de regiones ocurre en el procesamiento de imágenes. Se da un arreglo entero `image[1:n,1:n]`, donde cada entrada representa la intensidad del pixel. Los vecinos de cada pixel son los cuatro pixels que están arriba, abajo, a la izquierda y a la derecha. Dos pixeles pertenecen a la misma región si son vecinos y tienen el mismo valor. Luego, una región es un conjunto maximal de pixels que están conectados y tienen todos el mismo valor.
> El problema consiste en encontrar todas las regiones y asignar a todos los pixels de cada región una etiqueta única. En particular sea `label[1:n,1:n]` una segunda matriz, y asuma que inicialmente `label[i,j]= n*i + j`. El valor final de `label[i,j]` deberá ser el máximo de las etiquetas iniciales de la región a la cual el pixel `[i,j]` pertenece.
>
> Escriba un *algoritmo paralelo de grilla* que calcule los valores finales de `label`. La computación deberá terminar cuando `label` no cambie.

Un *algoritmo paralelo de grilla* típico donde el paralelismo es una componente secuencial por elemento de grilla, se reproduce a continuación:

```
real grid[n,n], newgrid[n,n];
bool converged = false;

process Grid[i = 1 to n, j = 1 to n] {
        while (!converged) {
                newgrid[i,j] = F(neighbors(i,j));
                barrier();
                "calc_converged"
                grid[i,j] = newgrid[i,j];
                barrier();
        }
}
```

La barrera es la primitiva de sincronización de alto nivel, denominada también *inclusión mutua*, cuya computation proper fue dada en el ejercicio 5 del práctico 6.

Se pide:

- **Barrera**

  1. Derivar la barrera del contador, o la del contador utilizando N bits.
  2. Demostrar progreso del multiprograma obtenido.
  3. Mostrar a través de un ejemplo como utilizaría el Inv de la barrera para hacer demostraciones de no interferencia entre "fases" de la computación.
  4. Implementar la barrera en algún lenguaje imperativo que soporte la noción de *threads*[1]. Intente utilizar primitivas de sincronización con una clara semántica o bien recurra al *busy wait*, aclarando el grado de atomicidad que supone válido en su arquitectura[2].
  5. Un problema de las barreras es que en general requieren de una precondición concreta para todas las componentes que la forman. Por ejemplo, la barrera del contador necesita $n = 0$ como Pre. Cuando se pasa la primera barrera, el valor de n ha cambiado y esta suposición ya no es válida. Lamentablemente para que resulte válido que $n = 0$ antes de la siguiente barrera se requeriría de *otra* barrera. Utilice la técnica del *sense reversal* [1] para solucionar este problema[3].

- **Region Labeling Problem**

  1. Dar la Pre, Post y el un invariante del sistema.
  2. Demostrar que la Post implica que el *region labeling problem* está resuelto.
  3. Explique porque, para que el algoritmo sea correcto, no resulta necesaria más atomicidad que la lectura y escritura de valores escalares. También explique porque el algoritmo funciona correctamente (aunque la detección de terminación puede complicarse) si quitamos las barreras.
  4. Implemente el algoritmo y genere el suficiente no-determinismo en su manejador de *threads* como para que corridas distintas generen interleavings distintos.

Referencias

[1] Gregory Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming*, Addison Wesley, 2000

[2] Doug Lea, *Concurrent Java Programming*, Java Series - Addison Wesley, 1999

[3] Jayadev Misra, *Phase Synchronization*, Information Processing Letters, Vol. 38, pp. 81-85, 1991 (Corrigenda, 41, p. 59, 1992).

[4] Scot Oaks and Henry Wong, *Java Threads*, O'Reilly, 1998

[1] C+pthreads, Java, ConcurrentPascal, etc.

[2] Cuidado, este punto no es trivial!

[3] Otra solución es construir barreras que no requieran de ningún tipo de inicialización. Usted ya vio dicho multiprograma para $N = 2$. Es posible construir barreras de N a partir de barreras de 2 con dos técnicas, la de *butterfly* y la de *dissemination* [1].