Taller de programación concurrente declarativa

Harold Armando Achicanoy Estrella

Octubre 8, 2017

Punto 1

El primer paso consiste en definir las variables declarativas a crear `{A, B, C, D}`. Cada una de ellas se creará en un hilo de forma independiente. No obstante, para hacer la evaluación de los hilos surge la necesidad de la sincronización, debido al hecho que para ligar las variables `{B, C, D}` con sus respectivos valores es necesario haber definido previamente la variable `A`.

Dado que los hilos se ejecutan de forma intercalada, no es posible saber cuál de ellos se va a ejecutar de primero, sin embargo, los hilos que se ejecuten y necesiten de la ligadura previa de ciertas variables para la realización de los cálculos, se suspenderan hasta que las variables necesarias se liguen en los otros hilos.

Por esta razón, los hilos que tiene a cargo la evaluación de las variables `{B, C, D}` quedan suspendidos y tienen que esperar a que el hilo que defina `A` termine con la asignación `A=1`. Una vez la variable `A` se encuentre ligada se evaluan los hilos que crean las variables `B`, `C` y `D`, en este orden respectivo.

Punto 2

Literal a)

La ejecución de la instrucción se suspende debido a que la variable declarativa `A` no ha sido ligada. Cuando se aplica la función `Filter` sobre la lista, la función no tiene problemas para evaluar los dos primeros elementos de la lista dado que son numéricos. Sin embargo, cuando se aplica al tercer elemento de la lista se presenta una suspensión de la función debido a que el argumento `case` no puede ejecutarse sobre una variable no declarada cuando se hace la evaluación `A>2`. Esto sucede bajo el modelo declarativo sin la adición de concurrencia en el modelo.

Literal b)

Agregando concurrencia al modelo con la creación de dos hilos, al ejecutar el código del presente literal se generan dos posibles resultados. Uno que es `Sal<optimized>` y el otro `5|_<optmized>`. Esto se debe a que la ejecución de los dos hilos creados es independiente y cualquiera de los dos puede ser ejecutado de primero.

El resultado `Sal<optimized>` se presenta cuando el hilo que toma el comando `{Show Sal}` es el primero en ser evaluado, esto ocurre dado que la ejecución se los hilos se hace de forma intercalada y hasta el momento de la ejecución del comando `Show` no se ha alcanzado a evaluar la función `Filter` sobre la lista en el otro hilo.

Por otro lado, el resultado `5|_<optmized>` se presenta cuando el hilo que tiene asociada la instrucción `Sal={Filter [5 1 A 4 0] fun {$ X} X>2 end}` es el primero en ejecutarse y no alcanza a intercalarse con el hilo que tiene el argumento `Show` hasta que se presenta la suspensión del primer hilo.

Literal c)

Cuando se ejecuta el código del presente literal únicamente se presenta el siguiente resultado: `5|_<optmized>`. Esto ocurre dado que el hilo que tiene asociada la instrucción `Sal={Filter [5 1 A 4 0] fun {$ X} X>2`

`end}` se ejecuta completamente sin tener que intercalarse con el hilo que tiene la instrucción `{Show Sal}`.

Literal d)

La ejecución de la presente instrucción produce como resultado: `[5 6 4]`. Aquí se cuenta con la ejecución de tres hilos. El primero que tiene a cargo la aplicación de la función `Filter` a la lista y evaluar la condición `X>2` sobre cada elemento. El segundo hilo encargado de ligar la variable declarativa `A` con el valor `6` como variable de flujo de datos. Y el tercer hilo encargado de mostrar el resultado del primer hilo.

De igual manera, los hilos se ejecutan de forma intercalada, alternando entre la aplicación de la función `Filter` sobre la lista definida, la ligadura de la variable `A` y la aplicación de la condición `X>2`. Es importante resaltar la actuación de la variable `A` como variable de flujo de datos, la cual permite continuar con la ejecución del primer hilo una vez ligada, cuando este se encuentra suspendido.

Finalmente la ejecución del tercer hilo que tiene el argumento `Show` se retrasa 1 segundo, para que los dos primeros hilos puedan alternarse el trabajo y finalmente se muestra el resultado del primer hilo un segundo después.

Punto 9

Funcionamiento del programa

La impresión de los carácteres `x` y `y` en el Browser de manera inmediata, se produce por el hecho de ser las primeras variables en evaluarse (`X`, `Y`) debido a que el parentesis enfatiza sobre su calculo inicial.

Por otro lado, la razón por la cual la suma `(X+Y)+Z` se retrasa 15 segundos en imprimir el resultado en pantalla es que para calcular la suma total, primero se debe evaluar el valor del parentesis donde se toma la suma de los tiempos máximos de retraso asociados, de este modo, el retraso máximo entre `X` y `Y` es de 6 segundos, posteriormente cuando se evalua `Z`, el tiempo de retraso asociado con `Z` es de 9 segundos.

Qué pasa si reemplazamos `(X+Y)+Z` por `X+(Y+Z)` o por `thread X+Y end + Z`?

Cuando se reemplaza por `X+(Y+Z)` se muestran en el `Browse` los carácteres `y`, `z` de manera inmediata, mientras la evaluación de `x` se tarda 9 segundos, valor que corresponde al máximo retraso entre la ejecución de `Y` y `Z`. Y tres segundos después se muestra el resultado 6.

Por otro lado, cuando se ejecuta la impresión de `thread X+Y end + Z`, los carácteres `x`, `y` y `z` se imprimen de manera inmediata en el Browse. Mientras el cálculo de la suma se tarda 9 segundos, valor correspondiente al máximo retraso de las tres funciones en comparación. Este comportamiento se presenta debido al hecho que el cálculo de la suma `X+Y` se genera a través de un hilo (cuyo tiempo máximo de ejecución es de 6 segundos), mientras en un hilo aparte se realiza la ejecución de `Z` con un tiempo de ejecución de 9 segundos.

En qué forma se presenta más rápido el resultado final?

El modo que genera la ejecución más rápida del resultado final es la inclusión de concurrencia en el modelo a través de la generación de dos hilos `thread X+Y end + Z`, permitiendo el cálculo independiente de los procesos involucrados en la suma total.

Cómo se programaría la suma de n números enteros $i_1, ..., i_n$ sabiendo que el entero i_j estará disponible sólo después de t_j ms, de manera que el resultado final se produzca lo más rápido posible?

Bajor el supuesto que tanto los enteros i_j como sus tiempos asociados t_j están ordenados en orden ascendente, bastaría con la creación de dos hilos, el primero que sume los enteros $i_1, ..., i_{n-1}$, mientras el otro hilo estaría encargado de evaluar i_n que es el que tiene mayor retraso asociado. De esta manera, el tiempo de cálculo empleado para la suma sería de t_n ms.