Criterios de Cobertura para Pruebas de Programas Paralelos

Emilio Hernández
emsca!usb!emilio@sun.com
Departamento de Computación y Tecnología de la Información
Universidad Simón Bolívar
Apartado 89000, Caracas 1080-A, Venezuela

Alejandro Teruel
emsca!usb!teruel@sun.com
Departamento de Computación y Tecnología de la Información
Universidad Simón Bolívar
Apartado 89000, Caracas 1080-A, Venezuela

Resumen

En este artículo se propone un esquema de análisis de cobertura para pruebas de programas paralelos con comunicación síncrona. La orientación del criterio definido tiene como objetivo servir de apoyo al desarrollo de software paralelo sobre una plataforma de transputadores.

Keywords: Paralelismo, Transputadores, C, Pruebas de Programas, Complejidad Ciclomática

1 Introducción

La prueba de programas (*testing*) es una fase muy importante del ciclo de desarrollo de software. Con frecuencia el tiempo invertido en esta etapa por algunas organizaciones de elaboración de software se eleva a un 40% del tiempo total de desarrollo [8, pág. 467]. El desarrollo de software paralelo es más difícil que el de software secuencial porque debe controlarse la coordinación de los procesos. Se han elaborado algunas herramientas experimentales de depuración y prueba de programas paralelos [5], de alcances todavía limitados.

En particular, las pruebas de programas secuenciales de tipo caja blanca [7] difícilmente se adaptan a las necesidades de pruebas de programas paralelos, porque deben contemplar elementos externos a la estructura del programa, como son las comunicaciones. En este artículo se propone un criterio de cobertura para hacer pruebas de tipo caja blanca de programas paralelos, en particular para los que utilizan el paradigma MIMD con comunicación síncrona definido por el esquema *OCCAM* de los transputadores INMOS [3]. Este criterio constituye la base para elaborar una herramienta automatizada de ayuda a las pruebas y depuración de programas paralelos.

2 Criterios de cobertura para programas paralelos

2.1 Grafo de Caminos Paralelos

En este trabajo se usó un modelo particular simplificado de un programa paralelo. Este modelo se toma del paradigma *OCCAM* de ejecución de procesos secuenciales en transputadores, cuyos mecanismos de sincronización y comunicación, tomados en cuenta para el análisis que sigue, pueden verse con mayor detalle en [3].

El grafo de caminos de un módulo secuencial contiene un nodo por cada condicional simple (**vértice o nodo de decisión**) y un nodo por cada secuencia maximal de instrucciones no condicionales, tales como asignaciones o llamadas a funciones, que a lo sumo contengan una instrucción de comunicación (**vértice o nodo de acción**). Para esto, los bloques de instrucciones se dividen en subsecuencias que comiencen con *send* o *recv*, excepto posiblemente la primera subsecuencia. Las aristas representan relaciones de precedencia de ejecución entre nodos. Existen dos nodos especiales, *INICIO* y *FIN* que representan los puntos de entrada y salida del módulo. Un camino se entiende en el sentido tradicional que se le da en teoría de grafos, debiendo comenzar en *INICIO* y terminar en *FIN*.

Un grafo de caminos paralelos se define de forma similar al grafo de caminos secuencial, la diferencia es que se trata de un grafo no conexo, en el que cada componente conexa representa a un módulo secuencial. Cada componente conexa i tiene un vértice inicial $INICIO_i$

y un vértice final FIN_i.

2.1.1 Caminos de ejecución paralelos

Dado un módulo paralelo M y su grafo G de caminos asociado a q componentes conexas $G_1 \ldots G_q$, se define como camino de ejecución paralelo una q-upla $(C_1, C_2, \ldots, C_q)$, donde C_i es un camino de ejecución secuencial de la componente conexa G_i del grafo de caminos paralelos. De este modo, el conjunto total de caminos paralelos, que denotaremos como CP_M, es el producto cartesiano de los conjuntos de todos los caminos de los módulos secuenciales. En la figura 1 se muestra un módulo paralelo formado por dos módulos secuenciales y su grafo de caminos, con dos componentes conexas.

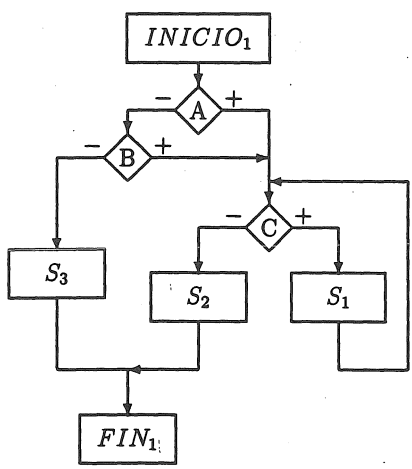
2.1.2 Conjuntos de cobertura paralela

Pueden definirse conjuntos de cobertura paralela de forma similar a los de los módulos secuenciales [7, 8]. Para un módulo paralelo M un conjunto de cobertura es un subconjunto de CP_M que cumple con ciertas condiciones. Se pueden definir varias clases de subconjuntos de CP_M.

- **Conjunto de cobertura paralela exhaustiva** ($CCPE_M$): el mismo CP_M.
- **Conjuntos de cobertura paralela de instrucciones** ($CCPI_M$): subconjuntos de CP_M que cubren todos los vértices de acción, es decir, cada vértice de acción es parte de algún camino de alguna q-upla perteneciente al conjunto.
- **Conjuntos de cobertura paralela de condiciones** ($CCPC_M$): subconjuntos de CP_M que cubren todas las aristas cuya primera componente es un vértice condicional, es decir, cada arista cuya primera componente es un vértice de decisión es parte de algún camino de alguna q-upla perteneciente al conjunto.
- **Conjuntos de cobertura ciclomática independiente** ($CCCicI_M$): subconjuntos de CP_M en los que se tiene una cobertura ciclomática [6] en cada una de las componentes conexas, es decir, en cada una de las dimensiones de las q-uplas del conjunto.

```
M1 () {
   if (A || B) {
      while (C) {
         /* S1 */
         Bloque;
      }
      /* S2 */
      send(M2, msg);
   }
   else {
      /* S3 */
      send(M2, msg);
   }
}
```

```
M2() {
   if (D) {
      /* S4 */
      recv(M1, msg);
   }
   else {
      /* S5 */
      recv(M1, msg)
   }
}
```



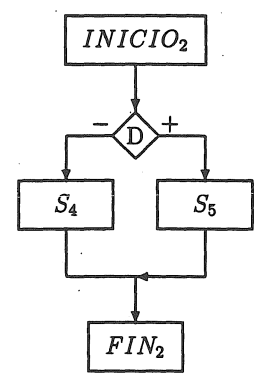


Figura 1: Módulo Paralelo y su Grafo de Caminos

2.2 Grafo de ejecución conjunta

Los conjuntos de cobertura descritos en la sección anterior, excepto $CCPE_M$, no contemplan una cobertura exhaustiva de las posibilidades de comunicación entre los módulos secuenciales. Si bien los tipos de cobertura garantizan la ejecución de todas las operaciones de comunicación al menos una vez, no se tiene un mecanismo que verifique si una instrucción de comunicación en un módulo secuencial se sincroniza exactamente con la que el programador desea, en otro de los módulos. Además, si una instrucción de comunicación puede eventualmente sincronizarse de varias maneras con distintas operaciones similares en otro módulo, no se ofrece hasta ahora ningún tipo de cobertura de tales actos de comunicación entre instrucciones de ambos módulos secuenciales. Por esta razón se ha establecido un esquema de fusión de los grafos de los módulos secuenciales en uno solo, denominado **grafo de ejecución conjunta**, a partir del cual se puede determinar ciertos tipos de cobertura. La elaboración del grafo de ejecución conjunta permite, además, realizar un análisis estático del módulo paralelo (en particular, verificación de compatibilidad de protocolos de comunicación) que puede arrojar información al programador acerca de posibles fuentes de error en el programa. La elaboración un grafo de ejecución conjunta se basa en el hecho de que existe una relación de orden parcial de ejecución entre las instrucciones de todos los módulos secuenciales. Esta relación está determinada por la ejecución de instrucciones de comunicación, que establecen un punto de sincronización entre módulos [4, 1].

El grafo de ejecución conjunta contiene como nodos a todos los que pertenecen a las componentes conexas, cumpliendo las siguientes propiedades:

- Los nodos de *INICIO* y *FIN* de cada componente conexa se combinan en uno sólo de *INICIO* y uno sólo de *FIN*, respectivamente.

- Un nodo perteneciente a una componente conexa puede tener como nodo sucesor a sus sucesores en la misma componente o a un nodo de otra componente conexa, siempre que lo permita la relación de orden parcial que existe en la ejecución de las secuencias de instrucciones representadas por los vértices.

- Todo nodo de comunicación debe estar acoplado (esto es, precedido o seguido) por un nodo de comunicación del módulo y el canal con el que establezca comunicación. La combinación de estos dos nodos de distintos módulos en uno solo dentro del grafo de ejecución conjunta se denomina *nodo de sincronización.*

- Puede haber réplicas de nodos de las componentes conexas en el grafo de ejecución conjunta, siempre que los subgrafos de caminos que continúan a partir de ellos sean distintos. Esto significa que no deben existir, por razones de simplificación, subgrafos idénticos dentro del grafo de ejecución conjunta.

Dado un conjunto de módulos secuenciales que conforman un módulo paralelo, generalmente existen muchos grafos de ejecución conjunta posibles. No obstante, es posible, dado un ordenamiento determinado de los módulos secuenciales, obtener un grafo de ejecución conjunta único, en el que los vértices cuyo orden de ejecución es indeterminado tienen en el grafo una relación de precedencia bien definida [2].

En la figura 2 se da un ejemplo de grafo de ejecución conjunta, construido a partir del módulo paralelo cuyo grafo de caminos independientes se da en la figura 1. En línea punteada están los nodos de sincronización.

3 Conclusiones

Se ha diseñado un criterio de cobertura para programas paralelos que permite realizar análisis estático y dinámico de programas. La construcción del grafo de ejecución conjunta puede dar información acerca de posibles interbloqueos *(deadlocks)*, al encontrarse con la necesidad de acoplar en algún camino dos vértices de comunicación del mismo tipo (*send* o *recv*) pertenecientes a distintos módulos. Por otro lado, una vez construido el grafo de ejecución conjunta puede aplicarse cualquiera de los criterios definidos para cobertura de módulos secuenciales, si consideramos a los vértices de sincronización como un vértice de acción normal y a los vértices repetidos los etiquetamos para diferenciarlos.

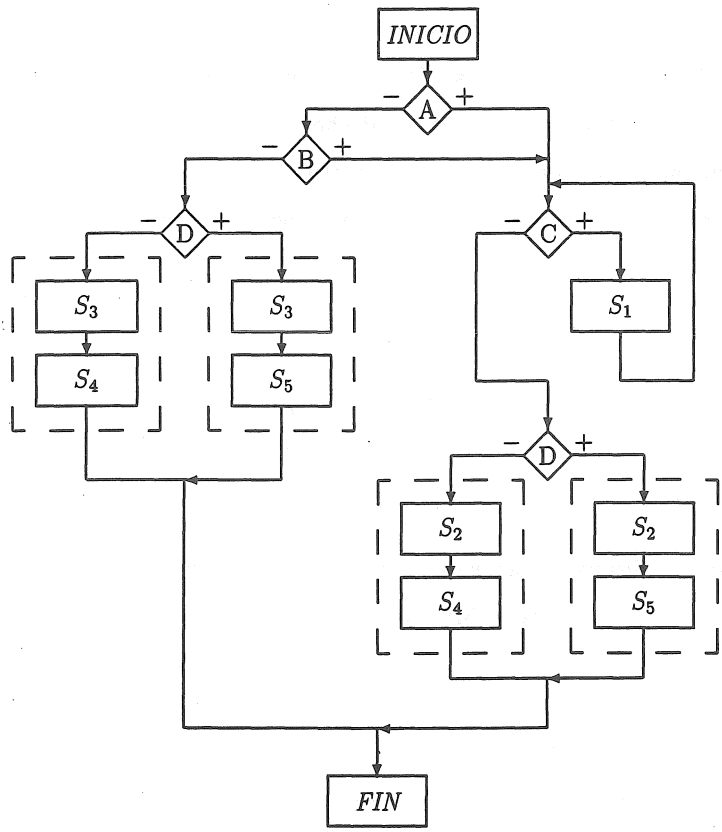


Figura 2: Grafo de ejecución conjunta del módulo de la figura 1

A partir de los criterios de cobertura definidos en este artículo se está diseñando actualmente un prototipo de una herramienta de ayuda a las pruebas de programas paralelos que en base a la instrumentación de los módulos para generar trazas, a la elaboración del grafo de ejecución conjunta, y al análisis *post-mortem* de dichas trazas, se determina el nivel de cobertura de las pruebas realizadas. Desde el punto de vista teórico se están realizando investigaciones para completar el esquema, añadiendo criterios para cobertura de ciclos y buscando simplificaciones cuando el módulo paralelo está formado por réplicas de módulos secuenciales idénticos (caso típico del esquema de programación *master-slave*).

Referencias

[1] C.J. Fidge. Partial orders for parallel debugging. In *Proceedings of the SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging.* SIGPLAN Not. (ACM), 1989.

[2] E. Hernández. Sistema de apoyo a las generación de datos de prueba para programas paralelos. Trabajo de Grado, Maestría en Ciencias de la Computación, Universidad Simón Bolívar, 1992.

[3] INMOS Limited. *Transputer development system*, 1988.

[4] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications ACM*, 21(7), 1978.

[5] M. Lutz. Testing tools. *IEEE Software*, mayo 1990.

[6] T. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4), 1976.

[7] G. Myers. *The Art of Software Testing.* John Wiley and Sons, 1979.

[8] R. Pressman. *Software Engineering.* McGraw-Hill, 1987.