PROGRAMACIÓN DE UN ALGORITMO PARALELO PARA LA OBTENCIÓN DE TESTORES

Mario Farías-Elinos[1], Patricia Rayón-Villela[2] y Manuel Lazo-Cortes[3]

[1]Laboratorio del Centro de Investigación, Universidad La Salle.
[2]Sección de Control, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN.
[3]Instituto de Cibernética, Matemáticas y Física, La Habana, Cuba.
e-mail: mfarias@sparcciulsa.ulsa.mx

RESUMEN

El presente trabajo muestra un programa que emplea técnicas de programación paralela, el cual permite realizar el cálculo de testores en un tiempo mucho menor que los algoritmos convencionales o seriados. Puesto que en este tipo de algoritmos se emplean cálculos con una complejidad exponencial, se consideran problemas NP completos (su solución se obtiene en un tiempo No Polinomial), característica que los hace aptos a ser resueltos aplicando técnicas de programación paralela.

ABSTRACT

The current paper shows a program which applies parallel programming techniques, which computes the typical test in less time than conventional or serial algorithms. This kind of problem is considered NP complete (problems which solve in Non Polynomial time), that's why we used a parallel programming technique in order to solve it.

INTRODUCCIÓN

Un testor se puede definir como el conjunto mínimo de rasgos o variables con los cuales se puede distinguir un objeto de cualquier otro dentro de un problema de clasificación supervisada.

En un inicio se le conoció bajo el nombre de *test* (*Prueba*), término que fue introducido por Cheguis y Yablonskii en 1955 (1), en la ex Unión de Repúblicas Socialistas Soviéticas, quienes se dedicaron a realizar pruebas de interruptores eléctricos para detectar las fallas; posteriormente Zhuravliov (2) en 1965 dio una formalización matemática, creando el término de *Testor*, lo que permitió considerar este concepto dentro del terreno de reconocimiento de patrones.

El problema de encontrar los testores puede ser comparable a dividir un número primo entre los valores positivos menores que él: es evidente que entre más grande sea el valor primo, más tiempo se requerirá para dividirlo entre los valores positivos menores.

Para obtener la solución a este problema se requerirá de un tiempo exponencial, debido a que en el cálculo de testores, cada variable i existente implica tener que realizar 2^i cálculos adicionales, lo cual ocasiona que el algoritmo tenga una complejidad $O(2^n)$ y que el tiempo de solución se incremente en forma exponencial.

PROCESAMIENTO PARALELO

El procesamiento paralelo implica tener dos o más procesadores trabajando en forma conjunta sobre un mismo problema, de manera que sea posible la reducción de su tiempo de solución.

La técnica de programación paralela tiene dos objetivos principales:

1. Reducir el tiempo de procesamiento utilizado por un algoritmo convencional.

2. Reducir la complejidad de un algoritmo.

El primero se cumple al momento de paralelizar, pero para el segundo se requieren técnicas de análisis de algoritmos paralelos.

Para llevar a cabo la paralelización hay que tomar en cuenta varias limitaciones, una de las cuales es la independencia de datos; es decir, que en un momento determinado no se esté accesando la misma localidad de memoria en forma simultánea, o bien, que no se dependa de un dato inmediato anterior.

Otro aspecto a considerar es la forma en que se dividirá el problema; para ello es necesario conocer la forma en que se maneja la información.

Dentro del procesamiento paralelo existen dos paradigmas (3):

1. Homoparalelismo: Se refiere a dividir el trabajo realizado por un algoritmo en subtareas idénticas y de igual carga, las cuales se ejecutan en forma simultánea e independiente (ver Figura 1).

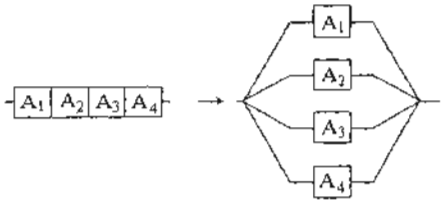

Figura 1. Homoparalelismo

2. Heteroparalelismo: Se refiere a dividir el trabajo realizado por un algoritmo en diferentes subtareas, pudiendo ser éstas de distinta carga, las cuales se ejecutan en forma simultánea e independiente (ver Figura 2).

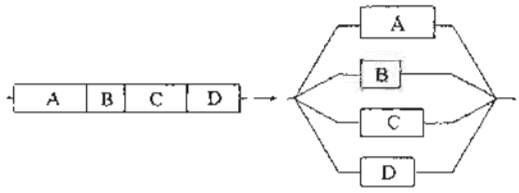

Figura 2. Heteroparalelismo

DESARROLLO

El algoritmo se programó en lenguaje C con librerías de PVM (*Parallel Virtual Machine* - Máquina Paralela Virtual) (4), sistema con el cual es posible comunicar varias computadoras UNIX y trabajar con su procesadores como si existiera una única computadora con varios procesadores.

Los testores, y particularmente los testores típicos, son un útil instrumento para la selección de variables en un problema de clasificación supervisada (5). Éste es un problema de complejidad exponencial, para el cual se han dado diferentes algoritmos que pretenden simplificar la búsqueda. Aún así, todos los algoritmos existentes hasta hoy son seriados y su eficiencia en algunos casos, para matrices de ciertas dimensiones deja mucho que desear. Es por eso que se desarrolló un algoritmo paralelo que disminuye los tiempos de cálculo; éste está basado en el algoritmo CC-difuso, descrito en 1980 (6) y que permite encontrar tanto los testores típicos clásicos como los difusos de Goldman.

La solución del problema se programó bajo el paradigma de homoparalelismo, el cual indica que cada procesador existente realiza el mismo trabajo; la diferencia es que cada procesador calcula diferentes testores, con lo que se logra reducir el tiempo de cómputo.

La idea, dentro del campo de reconocimiento de patrones, es encontrar las características mínimas necesarias para poder diferenciar un objeto de otro, por ejemplo:

Para reconocer un rostro podemos considerar las siguientes características:

- Tipo de ojos
- Color del ojo

- Tipo de ceja
- Tamaño de la boca
- Tamaño de los labios
- Tipo de nariz
- Color del pelo
- Forma del mentón
- Tamaño de la frente

Todas éstas son características que se toman en cuenta para poder diferenciar un rostro de otro, pero implica realizar comparaciones exhaustivas, y por lo tanto el proceso a nivel cómputo se hace lento e impráctico. Una forma de resolver este problema es encontrar un subconjunto que contenga un mínimo de características necesarias que permitan diferenciar un rostro de otro; a este subconjunto se le conoce como *testor típico*.

El cálculo de este subconjunto se lleva a cabo a través del una búsqueda exhaustiva, ya que se calculan todas las posibles combinaciones para formar al testor típico.

Estas características pueden agruparse en forma matricial de la siguiente manera:

	X_1	...	X_n
O_1	$X_1(O_1)$	...	$X_n(O_1)$
.	.		.
.	.		.
.	.		.
O_m	$X_1(O_m)$	...	$X_n(O_m)$

donde:

O_m representa un objeto.

X_n representa las características del objeto.

A esta matriz se le denomina *matriz de Aprendizaje* (*MA*).

A partir de la MA se realizan ciertos criterios de comparación, los cuales son diferentes para cada problema, y se obtiene la *Matriz de Diferencias (MD)*, la cual nos indica qué tanto se asemeja un objeto a otro.

Posteriormente, de la MD se eliminan los renglones repetidos y se verifica si hay alguna super-fila. El criterio a utilizar será: utilizando dos renglones *p* y *t* cualquiera, se localiza el primer cero en *p* que aparezca en la misma posición en *t*; si antes de éste se puede encontrar en el renglón *p* un uno en la misma posición que en el renglón *t* un cero, se dice que el renglón *p* es superfila del renglón *t*. Por ejemplo:

$p = <11101>$ y $t = <10001>$

Se puede observar que en la posición 4 de ambos renglones se tiene un valor de cero y que antes de éste existen en la posición 2 y 3 un valor uno en *p*, y cero en *t*, lo que implica que *p* es superfila de *t*, o bien, *t* es subfila de *p*.

Una vez que se realiza este proceso, se tiene como resultado una matriz denominada *Matriz Básica* o MB, con la cual podemos inicial el cálculo de testores con el algoritmo que se menciona más adelante.

Para que un subconjunto de características pueda ser considerada testor, no debe existir algún renglón que tenga valores de cero en las columnas de las características que formen el subconjunto.

El algoritmo utilizado es (7):

Para cada procesador existente:

Paso 1: Se busca el siguiente valor diferente de cero disponible (Pivote) en la MB.

Paso 2: Se agrega el Pivote a la lista del candidato a testor; posteriormente se verifica si el Pivote es un testor, y si lo es, se añade a la lista de testores y se salta al paso 5; de no ser así se sigue al paso 3.

Paso 3: Se obtiene una lista de los renglones causantes por los cuales el Pivote no fue testor y se inicializa a 1 el valor de j.

Paso 4: Para el j-ésimo renglón causante:

Paso 4.1: Se busca el primer valor diferente de cero que sea compatible con los valores en la lista del dato a testor; si es compatible, se agrega a la lista de candidato y se salta al paso 4.2.; de no ser así, se salta al paso 4.4.

Paso 4.2: Se verifica si el candidato es un testor. Si es testor, se agrega a la lista de testores y se brinca al paso 4.4. de no ser así se busca en

los renglones culpables a los nuevos culpables y se pasa en forma recursiva al paso 4.

Paso 4.3: Se obtiene de la lista de candidatos el valor más reciente y se brinca al paso 4.5.

Paso 4.4: Se busca en el j-ésimo renglón otro valor diferente de cero y se brinca al paso 4.2

Paso 4.5: Se incrementa el valor de j.

Paso 5: Se obtiene el siguiente 1 disponible en MB y se brinca al paso 2.

Los subpasos de 4 se realizan en forma recursiva, haciendo posible guardar la lista de renglones y pasar al siguiente renglón sin tener que volver a crear la lista.

Las únicas variables compartidas son: la que indica cuál es el siguiente Pivote disponible y los apuntadores de la lista de testores. Para evitar los problemas de acceso simultáneo por parte de dos o más procesadores a dichas variables, la implementación del algoritmo se realiza a través de semáforos, evitando así caer en *dead-lock* o *candado mortal*.

En el momento de llevar a cabo la concatenación de la lista se verifica que el nuevo testor aún no haya sido incluido: de ser así, se agrega, y de lo contrario, se desecha. Esto es con el fin de no tener testores repetidos, ya que este algoritmo encuentra todos los testores y sus permutaciones.

La idea de verificar si es testor se encuentra en la lista de renglones culpables, es evitar comparaciones innecesarias que incrementen la complejidad, además de que se garantiza que únicamente se encontrarán nuevos culpables en los renglones ya culpables.

RESULTADOS

Se instaló el PVM (4) en 3 computadoras UNIX; en una SPARCstation-20, en una SPARCclassic (éstas dos con procesadores SuperSPARC v.8) y en una AlphaServer 4100 (con dos procesadores Alpha AXP); lo cual permite trabajar con 4 procesadores en paralelo.

Uno de los problemas en donde existen un gran número de características para llevar a cabo la clasificación de objetos fue la identificación de diferentes herramientas de trabajo, tales como: tijeras, cuchillos, trinches, palas, entro otras; las cuales son seleccionados para ser depositados en diferentes cajas. Este problema tiene 132 características en total, las cuales al momento de realizar una comparación exhaustiva hacían que el proceso fuera lento y que en algún momento dado se perdieran herramientas o que se cayeran de la banda de transportación.

La idea de introducir el término de *testores* es obtener un subconjunto con el mínimo de características que permitan reducir el tiempo de comparación y evitar que se pierdan objetos o que se caigan de la banda de transportación y por lo tanto, se agilice el proceso.

El segundo problema donde se utilizó esta técnica fue dentro del campo de la medicina, para detectar cuáles son los factores que intervienen en el problema de la lactancia a menores; este problema tiene una matriz con 60 características, las cuales para llevar a cabo una clasificación generaba un número muy grande de comparaciones. Aquí también se aplicó el uso de testores, con lo cual ffue posible simplificar el número de comparaciones y clasificar de una manera más rápida los factores que provocaron los problemas de lactancia.

Se pudo observar que el tiempo del cálculo de testores se redujo considerablemente en un 65% aproximadamente.

No. de variables	80486	SPARC v.8	PVM
60	168 hrs.	132 hrs.	86 hrs.
132	584 hrs.	456 hrs.	298 hrs.

CONCLUSIONES

Uno de los problemas que existen para el cálculo de testores, es que los algoritmos existentes hasta el momento trabajan bajo la

filosofía de serialización y no todos son paralelizables, entre ellos podemos mencionar el BT y el REC (5).

Uno de los problemas que se presenta en el momento de programar PVM es que no se tiene control de la carga de trabajo que exista en ese momento en el procesador.

Un factor ventajoso en el uso del PVM es que no existe dependencia de la computadora, es decir, no se están utilizando instrucciones propietarias del procesador, y por lo tanto el código es transportable a cualquier computadora y se pueden utilizar todos los procesadores disponibles en la red que tengan PVM.

Existe un acceso libre para el uso del PVM a través de Internet, del cual es posible encontrar una versión beta para Windows™ de 32 bits.

Se observa que la técnica de paralelismo constituye una herramienta útil para solucionar problemas NP-completos.

Se sugiere que utilizar procesadores *RISC* debido que el tiempo de cómputo es mucho menor debido a sus características, lo cual agrega un valor adicional a las técnicas de paralelización.

REFERENCIAS

1. Chegis, I.A. y Yablonski, S.V. Acerca de los test para esquemas eléctricos; [en ruso] *Uspieji Matematicheskij Nauk*, T

2. Dmitriev, A. N., Zhuravliov, Yu. I. y Krendelev, F.P. Acerca de los principios matemáticos de la clasificación de objetos y fenómenos; [en ruso] Colección Análisis Discreto, T. 7, pp. 3-15, 1966, Novosibirsk.

3. Bauer Barr, E. *Practical Parallel Programming*, USA. Academic Press, Inc., 1992.

4. PVM: *Parallel Virtual Machine, A Users Guide and Tutorial for Networked Parallel Computing*, Boston, MIT Press, 1994.

5. Ruíz Shulcloper, J., Alba Cabrera, E. y Lazo Cortés, M. *Introducción a la teoría de testores*. México. Serie verde, Depto. Ing. Eléctrica CINVESTAV, 1995.

6. Goldman, R.S. Problemas de la teoría de los testores difusos. *Avtomatika I Telemejanika.*, No. 10, pp 146-153, 1980.

7. Farías-Elinos, M., Rayón-Villela, P. y Lazo-Cortés, M. Cálculo de Testores utilizando un Algoritmo Paralelo, XXIX Congreso Nacional de Matemáticas, San Luis Potosí, S.L.P., del 6 al 12 de Octubre de 1996.