Estructura y Tecnología de Computadores

Práctica de Ensamblador 2006/2007

Introducción

El objetivo de esta práctica es realizar un estudio de optimización del ensamblador de la familia x86. Para ello se trabajará sobre un problema concreto, el de encontrar ciclos en una clase particular de grafo. Se tendrá que codificar inicialmente el algoritmo en ensamblador del 8086 exclusivamente, y se introducirán instrucciones de la familia IA-32 (Intel Architecture-32 bits, es decir, 386 en adelante) para mejorar el rendimiento. El programa constará de una rutina principal en C y subrutinas en ensamblador con las características anteriores, desarrollados en el entorno Borland C++ 3.1.

Resolver un problema de una forma eficiente consiste en elegir algoritmos y estructuras de datos que desde un punto de vista de alto nivel nos garanticen una cierta eficiencia teórica. En el estudio de la complejidad computacional de un algoritmo, se intenta caracterizar el tiempo de ejecución para toda una clase de problemas, en función de cierto parámetro *n* particular (el número de elementos en una lista, el número de nodos en un grafo, etc.). En problemas tratables, la expresión suele ser polinómica en *n*, ignorando frecuentemente constantes multiplicativas, puesto que para un *n* suficientemente grande, la influencia del *n* domina sobre el valor particular que tengan esas constantes. Son éstas últimas las que podemos variar según la eficiencia de la implementación de nuestro algoritmo (se incluye aquí tanto la estructura del procesador, como el compilador o nuestra habilidad para optimizar el código incluso en ensamblador). En problemas con el *n* fijado o acotado a un rango, a menudo esas constantes juegan un papel importante, incluso frente a algoritmos con expresiones de complejidad en *n* muy distintas.

La optimización del código es un problema complejo en el que intervienen las características de la arquitectura del procesador y del resto de subsistemas que componen el ordenador. Los dos aspectos principales que nos interesarán son:

-La estructura del procesador y correspondientemente del juego de instrucciones, es decir, de la habilidad de codificar el algoritmo de forma que se usen eficientemente los recursos de cada repertorio de instrucciones.

-La eficiencia del subsistema de memoria, que normalmente está formado por la memoria principal y la cache. Un programa con gran localidad de referencia tendrá una mayor probabilidad de ejecutarse y operar con datos desde la cache en lugar de la memoria principal, mucho más lenta.

El problema a resolver

El problema a resolver es, dado un grafo dirigido de una clase especial, clasificar sus nodos indicando a qué ciclo está asociado cada nodo. Este grafo tiene la particularidad de que cada nodo posee un único sucesor. Una representación útil de esta clase de grafos se reduce simplemente a un vector con un elemento por cada nodo donde se anota su sucesor (en lugar de considerar matrices de incidencia o de adyacencia) al que llamaremos **vector de sucesores**. Si hacemos un recorrido de este grafo partiendo de un nodo arbitrario acabaremos cayendo en un ciclo, puesto que todos los nodos tienen sucesor. Se pretende identificar para cada nodo el ciclo al que pertenece o al que finalmente van a parar sus sucesores. El algoritmo para encontrar un ciclo consiste entonces en escoger un nodo de prueba inicial y buscar entre los sucesores un nodo ya visitado. Puesto que todos los nodos tienen un sucesor, tarde o temprano encontraremos un nodo ya visitado. Como se indicó, a cada nodo se le puede asociar un ciclo, bien porque el nodo pertenece al ciclo o bien porque es el ciclo al que se llega siguiendo el camino de sucesores del nodo.

El objetivo final es identificar el ciclo a que corresponde cada nodo indicando además si pertenece al ciclo o no. Para ello a la rutina en ensamblador se le pasan tres parámetros según el siguiente prototipo.

```
clasif_nodos(int n, int * ptgrafo, int * ptresul);
```

El primero es un entero *n* con el número de nodos del grafo. El segundo es un puntero que apunta al vector de sucesores que describe el grafo (éste se ha ubicado previamente por el C y se ha creado el grafo en él). El tercero también es un puntero a un vector con n elementos enteros (uno por nodo) ubicado previamente por el C e inicializado a cero, que nos servirá como marca de nodo no etiquetado. Cada elemento se usa para indicar la etiqueta del ciclo asociado al nodo. Si han aparecido *c* ciclos (*c* debe ser menor o igual que *n*) se anotará en cada elemento un número de 1 a *c* que indica el ciclo. Para señalar también si cada elemento es parte del ciclo o sólo lleva a él, se usa el bit más significativo del número, poniéndolo a uno si es del ciclo o cero en caso contrario. El algoritmo deberá funcionar para grafos con número de nodos *n* desde 1 hasta 500. Se podrán usar variables globales para hacer cálculos o mantener estados intermedios en la rutina en ensamblador. Para encontrar todos los nodos que llevan a un ciclo particular, miraremos todos los predecesores de los elementos del ciclo y sucesivamente sus predecesores hasta no encontrar ningún predecesor más, o como alternativa mas sencilla, examinamos cada nodo no etiquetado, adjudicando a el y sus sucesores una etiqueta nueva de clase que sería definitiva si cerramos un ciclo encontrando un sucesor con la misma etiqueta o bien adjudicando a todos los explorados la etiqueta preexistente del nodo con otra etiqueta al que lleguemos. Para plantear un algoritmo completo conviene dibujar un grafo concreto como ejemplo (que sea lo más general posible cumpliendo las restricciones del problema) y razonarlo sobre él.

La medida del tiempo

A partir del procesador Pentium, Intel introdujo la instrucción RDTSC (Read Time Stamp Counter). Esta instrucción permite leer un contador de 64 bits que se incrementa con cada ciclo de reloj de la máquina. La cuenta se devuelve en la pareja de registros EDX y EAX. Esto quiere decir que tendremos una cuenta del número de ciclos que tarda nuestra subrutina si leemos el contador al principio y al final de su ejecución y restamos ambas medidas (se incluye ejemplo de este código en ensamblador/C). Para usar esta instrucción hay que tener en cuenta dos detalles:

1) El ensamblador del Borland C 3.1 es anterior a la introducción de esta instrucción, por lo que habrá que incorporarla en nuestro código explícitamente con su opcode mediante una sentencia ‘db’ (*define byte*)

```
...
db 0fh, 31h  ; opcode de la instrucción RDTSC
...
```

Este código también se puede declarar en una macro (llamándola por ejemplo ‘rdtsc’) lo que daría más legibilidad al programa.

2) Esta instrucción tiene una imprecisión de base, porque los procesadores modernos poseen *pipelines* muy profundos en los que se encuentran en proceso varias partes de instrucciones diferentes simultáneamente, o poseen características superescalares por las que pueden estar ejecutando varias instrucciones a la vez. Así, no se garantiza que cuando se lea el contador se hayan acabado de ejecutar las instrucciones precedentes ni tampoco que no haya comenzado previamente la ejecución de las siguientes. Por ejemplo, en un caso extremo de Pentium IV, es posible tener alrededor de cien instrucciones en algún punto de su ejecución. Para evitar en parte esto, Intel recomienda usar una instrucción “serializante” (*serializing*) antes, para garantizar que todas las instrucciones anteriores hayan acabado. Una instrucción serializante es la CPUID (opcode 0fh a2h, disponible a partir de los últimos 486). Esta instrucción se usa para proporcionar información muy variada sobre el procesador (identificación, características soportadas, tipos y tamaños de caché, etc), que se devuelve en EAX, EBX, ECX, y EDX y no posee otros efectos aparte de los mencionados. Usar esta técnica si se cree necesario.

Elaboración de la práctica

Para la subrutina en ensamblador, se deberá lograr primero una implementación en código 8086 exclusivamente. La introducción de código 386 se hará con el criterio de mejorar su tiempo de ejecución. Documentar y razonar en la medida que se pueda los cambios que se realicen y sus tiempos también es una parte de la práctica. Hay que insistir en que el objetivo no es tanto lograr los mejores tiempos cambiando totalmente la rutina 8086 por código 386, sino identificar **pequeños cambios** que mejoren en algo los tiempos. Podemos utilizar el *profiler* del Borland C++ para comprobar como podemos optimizar el código o bien guiarnos por nuestras propias medidas. El profiler es capaz de determinar el tiempo que pasa el programa ejecutando cualquier área o línea de código del mismo que nos interese, aunque para una medida de un intervalo de tiempo tan corto como comprobar la diferencia de tiempos debida a cambiar una instrucción por otra no es útil.

La fecha límite para la entrega de la práctica, salvo circunstancia excepcional, será de una semana antes de la entrega de actas (consultar con el profesor para conocer exactamente la fecha concreta). Para la corrección se deberán presentar los programas fuente en C o ensamblador listos para ensamblar y compilar de forma que se pueda producir con ellos el ejecutable final y los cambios documentados junto con un listado de los tiempos medidos (pueden estar incluidos en los propios ficheros fuentes).

Para cualquier duda, recurrir a las tutorías, o si es una cuestión concreta, al correo electrónico: jpineiro@ull.es.