Unidad

III: Optimización

Las optimizaciones pueden realizarse de diferentes formas. Las optimizaciones se realizan en base al alcance ofrecido por el compilador.

La optimización va a depender del lenguaje de programación y es directamente proporcional al tiempo de compilación; es decir, entre más optimización mayor tiempo de compilación.

La optimización es un proceso que tiene a minimizar o maximizar alguna variable de rendimiento, generalmente tiempo, espacio, procesador, etc.

3.1 Tipos de optimización

Dentro de los tipos de optimización se derivan los tipos de optimización local, optimización de ciclo, optimización global y optimización de mirilla.

3.1.1 Locales

La optimización local se realiza sobre módulos del programa. En la mayoría de las ocasiones a través de funciones, métodos, procedimientos, clases, etc.
La característica de las optimizaciones locales es que solo se ven reflejados en dichas secciones.

La optimización local sirve cuando un bloque de programa o sección es crítico por ejemplo: E/S, la concurrencia, la rapidez y confiabilidad de un conjunto de instrucciones. Como el espacio de soluciones es más pequeño la optimización local es más rápida. Como el espacio de soluciones es más pequeño la optimización local es más rápida.

3.1.2 Ciclos

Los ciclos son una de las partes más esenciales en el rendimiento de un programa dado que realizan acciones repetitivas, y si dichas acciones están mal realizadas, el problema se hace N veces más grandes. La mayoría de las optimizaciones sobre ciclos tratan de encontrar elementos que no deben repetirse en un ciclo.

Sea el ejemplo:

```
while(a == b) {
  int c = a;
  c = 5;
  …;
}
```

En este caso es mejor pasar el Int c =a; fuera del ciclo de ser posible.

El problema de la optimización en ciclos y en general radica es que muy difícil saber el uso exacto de algunas instrucciones. Así que no todo código de proceso puede ser optimizado. Otros uso de la optimización pueden ser el mejoramiento de consultas en SQL o en aplicaciones remotas (sockets, E/S, etc.)

3.1.3 Globales

La optimización global se da con respecto a todo el código. Este tipo de optimización es más lenta pero mejora el desempeño general de todo programa. Las optimizaciones globales pueden depender de la arquitectura de la máquina.

En algunos casos es mejor mantener variables globales para agilizar los procesos (el proceso de declarar variables y eliminarlas toma su tiempo) pero consume más memoria. Algunas optimizaciones incluyen utilizar como variables registros del CPU, utilizar instrucciones en ensamblador.

3.1.4 De mirilla

La optimización de mirilla trata de estructurar de manera eficiente el flujo del programa, sobre todo en instrucciones de bifurcación como son las decisiones, ciclos y saltos de rutinas.

La idea es tener los saltos lo más cerca de las llamadas, siendo el salto lo más pequeño posible.

Instrucciones de bifurcación

Interrumpen el flujo normal de un programa, es decir que evitan que se ejecute alguna instrucción del programa y salta a otra parte del programa.

Por ejemplo: el “break”

```
    Switch (expresión que estamos evaluando)
{
    Case 1: cout << “Hola” ;
    Break;
    Case 2: cout << “amigos”;
    Break;
    }
```

3.2 Costos

Los costos son el factor más importante a tomar en cuenta a la hora de optimizar ya que en ocasiones la mejora obtenida puede verse no reflejada en el programa final pero si ser perjudicial para el equipo de desarrollo. La optimización de una pequeña mejora tal vez tenga una pequeña ganancia en tiempo o en espacio pero sale muy costosa en tiempo en generarla.

Pero en cambio si esa optimización se hace por ejemplo en un ciclo, la mejora obtenida puede ser N veces mayor por lo cual el costo se minimiza y es benéfico la mejora.

Por ejemplo:

for(int i=0; i < 10000; i++); si la ganancia es de 30 ms 300s

3.2.1 Costo de ejecución (Memoria, registros, pilas)

Los costos de ejecución son aquellos que vienen implícitos al ejecutar el programa.

En algunos programas se tiene un mínimo para ejecutar el programa, por lo que el espacio y la velocidad de los microprocesadores son elementos que se deben optimizar para tener un mercado potencial más amplio.

Las aplicaciones multimedia como los videojuegos tienen un costo de ejecución alto por lo cual la optimización de su desempeño es crítico, la gran mayoría de las veces requieren de procesadores rápidos (e.g. tarjetas de video) o de mucha memoria. Otro tipo de aplicaciones que deben optimizarse son las aplicaciones para dispositivos móviles.

Los dispositivos móviles tienen recursos más limitados que un dispositivo de cómputo convencional razón por la cual, el mejor uso de memoria y otros recursos de hardware tiene mayor rendimiento. En algunos casos es preferible tener la lógica del negocio más fuerte en otros dispositivos y hacer uso de arquitecturas descentralizadas como cliente/servidor o P2P.

3.2.2 Criterios para mejorar el código

La mejor manera de optimizar el código es hacer ver a los programadores que optimicen su código desde el inicio, el problema radica en que el costo podría ser muy grande ya que tendría que codificar más y/o hacer su código más legible. Los criterios de optimización siempre están definidos por el compilador.

Muchos de estos criterios pueden modificarse con directivas del compilador desde el código o de manera externa. Este proceso lo realizan algunas herramientas del sistema como los ofuscadores para código móvil y código para dispositivos móviles.

3.2.3 Herramientas para el análisis del flujo de datos

Existen algunas herramientas que permiten el análisis de los flujos de datos, entre ellas tenemos los depuradores y desambladores. La optimización al igual que la programación es un arte y no se ha podido sistematizar del todo.