UNIDAD 3 "OPTIMIZACIÓN"

1. INTRODUCCIÓN

La optimización consiste en mejorar el código intermedio de modo que quede un código máquina más rápido de ejecutar.

Con esto también se obtiene optimización temporal y optimización espacial.

2. CAPITULO 1: TIPOS DE OPTIMIZACIÓN

¿QUÉ ES LA OPTIMIZACIÓN?

El objetivo de la optimización es mejorar código objeto final, preservando significado del programa. Es decir, realizar una serie de transformaciones al código intermedio para obtener un mejor rendimiento en cuanto a velocidad de ejecución entre otros parámetros. Aquí aplica que entre menos código basura o innecesaria se tenga más rápido funcionara nuestro programa

Existen diversas técnicas de optimización se pueden clasificar o dividir de diversas formas. Las siguientes son en función de la dependencia de la arquitectura:

- Dependientes de la maquina: técnicas que solo se pueden aplicar a una determinada maquina objeto.
- Independientes de la maquina: técnicas que son aplicables a cualquier maquina objeto.

Y a continuación vamos a describir en específico las que son en función de ámbito de la aplicación.

Sección 1.1: Locales

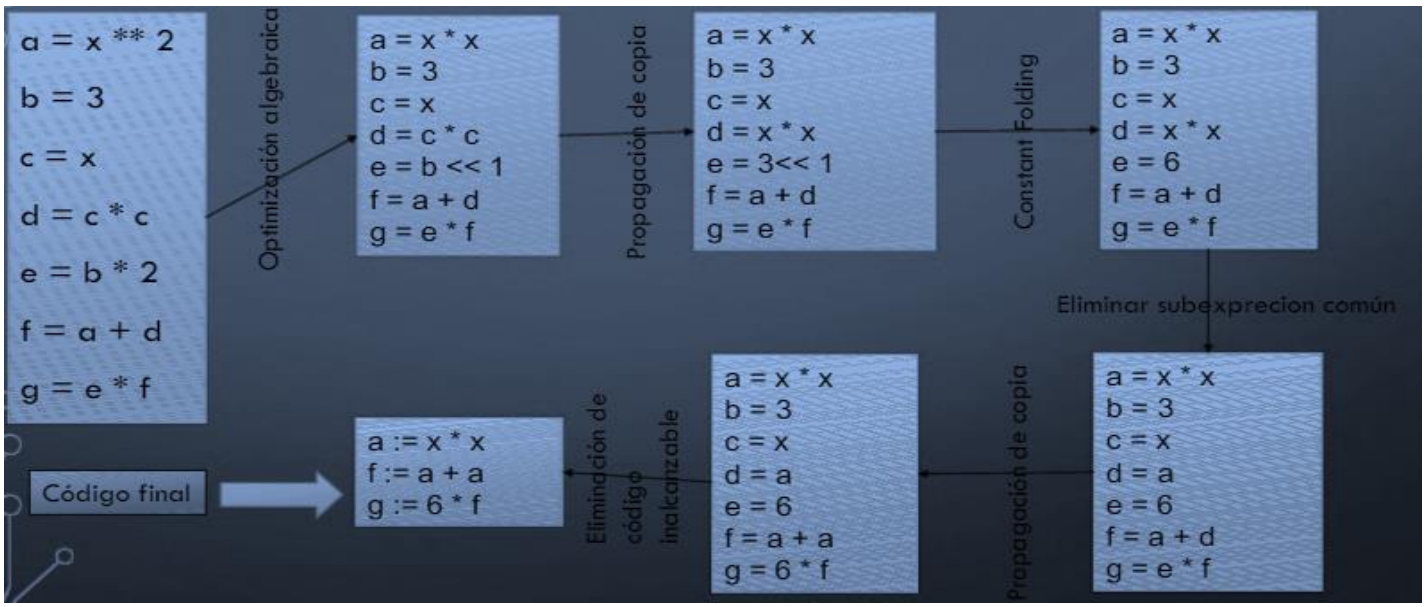
La optimización local se realiza sobre módulos del programa. En la mayoría de las ocasiones a través de funciones, métodos, procedimientos, clases, etc.

Las características 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: la 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.

Entre las diferentes optimizaciones locales encontramos:

- Simplificaciones algebraicas
- Propagación de copias y constantes.
- Constant folding
- Eliminación de subexpresiones redundantes o comunes.
- Eliminación de código incansable

EJEMPLO DE SECUENCIA DE OPTIMIZACIONES



Sección 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 en que es muy difícil saber el uso exacto de algunas instrucciones. Así que no todo código de proceso puede ser optimizado. Otro uso de la optimización puede ser el mejoramiento de consultas en SQL o en aplicaciones remotas (sockets, E/S, etc.).

Existen diferentes estrategias de optimización dentro de bucles (local o global):

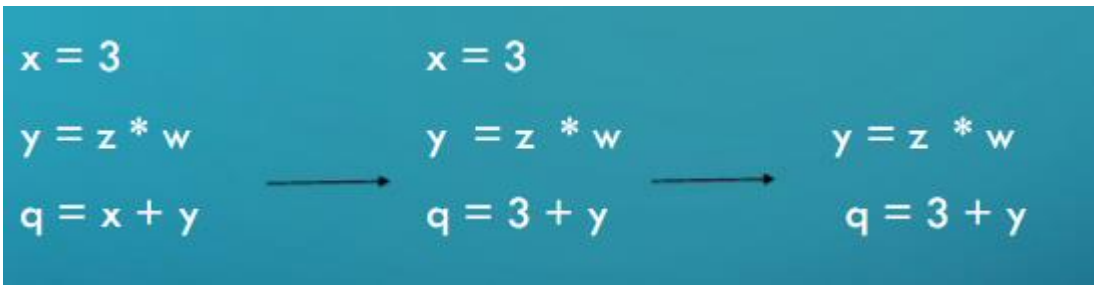
- Expansión de bucles (loop unrolling)
- Reducción de frecuencia (Frequency reduction)
- Reducción de potencia (strenght reduction)

Sección 1.3: Globales

Esta optimización es muy lenta pero mejora el desempeño general de todo el programa. Es necesario crear el grafo de flujo de ejecución de todo el código ya que este nos permite una mejor comprensión de las instrucciones y manejo de datos dentro del código.

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.

EJEMPLO



Ya que x = 3 corresponde a fuera del bloque básico o nodo del grafo simplemente aplicando teoría de las optimizaciones locales para esta que es global.

Sección 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.

Ideas básicas:

- Se recorre el código buscando combinaciones de instrucciones que pueden ser reemplazadas por otras equivalentes más eficientes.
- Se utiliza una ventana de n instrucciones y un conjunto de patrones de transformación (patrón, secuencias, remplazan).
- Las nuevas instrucciones son reconsideradas para las futuras optimizaciones.

Ejemplos:

- Eliminación de cargas innecesarias
- Reducción de potencia
- Eliminación de cadenas de saltos

3. CAPITULO 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

Sección 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.

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.

Sección 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.

Sección 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.

Depuradores

Un depurador es una aplicación que permite correr otros programas, permitiendo al usuario ejercer cierto control sobre los mismos a medida que se ejecutan, y examinar el estado del sistema (variables, registros, banderas, etc.) en el momento en que se presente algún problema.

El propósito final de un depurador consiste en permitir al usuario y observar y comprender lo que ocurre "dentro" de un programa mientras el mismo es ejecutado.

Desambladores

Es un programa de computador que traduce el lenguaje de maquina a lenguaje ensamblador, la operación inversa de la que hace el ensamblador.

La salida de un desensamblador, es a menudo formateada para la legibilidad humana en vez de ser adecuada para la entrada a un ensamblador, haciendo que éste sea principalmente una herramienta de ingeniería inversa.

4. CONCLUSION

La optimización no solo la desarrollan los programas y/o programadores, todos pueden participar dentro de una empresa desde el usuario final hasta los propios ejecutivos. El objetivo de las técnicas de optimización es mejorar el programa objeto para que nos dé un rendimiento mayor. La mayoría de estas técnicas vienen a compensar ciertas ineficiencias que son inherentes al concepto de lenguaje de alto nivel, el cual suprime detalles de la maquina objeto para facilitar la tarea de implementar un algoritmo.

5. CONCEPTOS

- Grafo: Es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto
- Bloque básico: Serie de instrucciones o sentencias consecutivas que el flujo de control lo recorre de principio a fin.
- Simplificaciones algebraicas: Son sentencias que pueden eliminarse o simplificarse.
- Propagación de copias: Optimización permite al programador utilizar variables como constantes sin introducir ineficiencias hasta la siguiente asignación de dicha variable.
- Constant folding (ensamblamiento): Consiste en remplazar las expresiones por su resultado cuando se pueden evaluar en tiempo de compilación.
- Código inalcanzable: códigos de bloques básicos que no son destino de ningún salto y por lo tanto pueden eliminarse.
- Mirilla: Secuencia corta de instrucciones objeto.

6. REFERENCIAS

Gómez, E.(2015). *Optimización*, de Academia Sitio web: http://www.academia.edu/29416690/lenguajes_y_automatas_ii_optimizacion

Tiumbra, H. (noviembre 24, 2014). *Lenguajes y Automatas 2 Optimización*, de SCRIBD Sitio web: https://es.scribd.com/document/248014105/Lenguajes-y-Automatas-2-Optimizacion

REPORTE

Como concepto general de optimizar hacemos referencia a buscar la mejor manera de realizar una actividad, y en este caso el optimizar el código es de gran utilidad para realizar las tareas de la forma más eficiente posible y esto se obtiene realizando unas transformaciones al código intermedio, al desechar líneas de código innecesarias o basura y obteniendo un código máquina optimizado.

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.

Las diferencias entre los distintos tipos de optimización que se definen, es que las locales reemplazan operaciones costosas de la máquina por otras menos costosas, y se realiza sobre bloques básicos. De ciclo es difícil saber el uso exacto de las instrucciones; aquí la idea es centrar la optimización en las partes más usadas y no en todo el programa. La global, pues como su nombre lo dice, corresponde a la optimización de todo el grafo de flujo de ejecución respecto a todo el código, es más lenta que las otras pero se obtienen mejores resultados. Y la de mirilla intenta rehacer el código de manera más eficiente distintos factores.

Y respecto a los costos, es de vital importancia tomarlo en cuenta ya que, a la hora de elegir algún tipo de optimización, influye mucho porque puede perjudicar el equipo de desarrollo y si no se toma la mejor decisión, se podrían obtener algunos resultados no esperados, además de que los costos pueden ser altos y no por la solución más viable.