Instituto Tecnológico de Nuevo León

Lenguaje y Autómatas II

Unidad 3

Jesús Alexis Jiménez Reyna N. control: 15480465

Carrera: Ing. Sistemas Computaciones

Ing. Juan Pablo Rosas Baldazo

N.L. Guadalupe 09 de noviembre del 2018

LENGUAJES Y AUTÓMATAS II

PARTE 1)

UNIDAD 3. 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.

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

Reacondicionamiento de operadores

Cambiar orden de evaluación aplicando propiedades conmutativa, asociativa y

distributiva.

V:= W * X *(Y + Z)

=

V := (Y + Z) * W * X

Código no optimizado, con 7 líneas de código.

1. MOV AX, W

2. MUL AX, X

3. MOV t1, AX

4. MOV AX, Y

5. ADD AX, Z

6. MUL AX, t1

7. MOV V, AX

Código optimizado a solo 5 líneas de código.

1.

2.

3.

4.

5.

MOV AX, Y

ADD AX, Z

MUL AX, W

MUL AX, X

MOV V, AX

3.1.2 Ciclos

Factorización de expresiones invariantes

Expresiones invariantes de bucle: expresiones cuyo valor es constante durante toda la ejecución del bucle

>incluyen constantes y/o variables no medicadas en el cuerpo del bucle

Principio: Mover expresiones invariantes desde el cuerpo hasta la cabeza del bucle

-> al sacarlas del bucle, pueden quedar dentro de otro bucle externo)

-> repetir proceso.

Ejemplos:

while (i

>

Algunas optimizaciones son difíciles de implementar

Algunas optimizaciones son costosas en términos de tiempo de compilación

La optimización más elaborada es difícil y costosa

Meta: Mejora Máxima con costo mínimo

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

3.1.3 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

• 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 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 crítica, 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 tiene 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.

3.2.2 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

Criterios de optimización

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

La optimización al igual que la programación es un arte y no se ha podido sistematizar del todo.

Existen algunas herramientas que permiten el análisis de los flujos de datos, entre ellas tenemos:

Depurador

Es una aplicación que permite correr otros programas, permitiendo al usuario ejercer cierto control sobre los mismos a medida que los estos 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 observar y comprender lo que ocurre "dentro" de un programa mientras el mismo es ejecutado.

En los sistemas operativos UNIX/LINUX, el depurador más comúnmente utilizado es gdb, es decir el depurador de GNU. Éste ofrece una cantidad muy extensa y especializada de opciones. Es muy importante entender el hecho de que un depurador trabaja sobre archivos ejecutables. Esto quiere decir que el mismo funciona de forma independiente al lenguaje en que se escribió el programa original, sea éste lenguaje ensamblador o un lenguaje de medio o alto nivel como C.

Desamblador

Es un programa de computadora que traduce el lenguaje de máquina a lenguaje ensamblador, la operación inversa de la que hace el ensamblador. Un desensamblador difiere de de un descompilador, en que éste apunta a un lenguaje de alto nivel en vez de al lenguaje ensamblador.

Diagrama de flujo de datos

Es una herramienta de modelización que permite describir, de un sistema, la transformación de entradas en salidas; el DFD también es conocido con el nombre de Modelo de Procesos de Negocios (BPM, Business Process Model).

Diccionario de datos

El Diccionario de Datos es un listado organizado de todos los elementos de datos que son pertinentes para el sistema, con definiciones precisas y rigurosas que le permite al usuario y al proyectista del sistema tener una misma comprensión de las entradas, de las salidas, de los componentes de los repositorios, y también de cálculos intermedios.

CONCLUSIONES:

En este trabajo se ha presentado la técnica de la optimización de código, como un medio de mejora del código objeto producido por un compilador. Dicha mejora va a ser evaluada en una reducción del tamaño de código objeto generado y, sobre todo, en una mayor velocidad de ejecución del programa objeto.

BIBLIOGRAFIAS:

http://itpn.mx/recursosisc/7semestre/leguajesyautomatas2/Unidad%20III.pdf

http://ditec.um.es/~jmgarcia/papers/ensayos.pdf

PARTE 2)

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

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.

GLOBALES: 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.

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

DICCIONARIO DE DATOS: El Diccionario de Datos es un listado organizado de todos los elementos de datos que son pertinentes para el sistema, con definiciones precisas y rigurosas.

DESAMBLADOR: Es un programa de computadora que traduce el lenguaje de máquina a lenguaje ensamblador, la operación inversa de la que hace el ensamblador. Un desensamblador difiere de un descompilador, en que éste apunta a un lenguaje de alto nivel en vez de al lenguaje ensamblador.

DEPURADOR: Es una aplicación que permite correr otros programas, permitiendo al usuario ejercer cierto control sobre los mismos a medida que los estos se ejecutan, y examinar el estado del sistema

PARTE 3)

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

Como el tiempo de optimización es gran consumidor de tiempo (dado que tiene que recorrer todo el árbol de posibles soluciones para el proceso de optimización) la optimización se deja hasta la fase de prueba final

Optimización de código: La optimización de código puede realizarse durante la propia generación o como paso adicional, ya sea intercalado entre el análisis semántico y la generación de código (se optimizan las cuádruplas) o situado después de ésta (se optimiza a posteriori el código generado).

Reordenación del código

En muchas máquinas, la multiplicación en punto fijo de dos operando de longitud 1 da un operando de longitud 2, mientras la división necesita un operando de longitud 2 y otro de longitud 1 para dar un cociente y un resto de longitud 1.

Optimización Local

• 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

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.

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.