INSTITUTO TECNOLÓGICO DE CHETUMAL

Actividad:

Integrar equipos, para analizar códigos intermedios existentes y proponer algunas mejoras. **Materia:**

Lenguaje y autómatas II

Maestro:

Miam Lopez Diego Aurelio

Alumno:

Guzman Zarrabal Julio Manuel

Nolasco Acosta Angel Alexis

Torres Guerrero Victor Fernando

Mukul Cab Andres Oswaldo

Chetumal Quintana Roo a 21 de abril del 2024

1. Optimización Local: Esta técnica se aplica a bloques de código que se ejecutan de principio a fin sin desviaciones. Por ejemplo, si tienes un bloque de código que realiza la misma operación aritmética varias veces, puedes calcular el resultado una vez y almacenarlo en una variable para su uso posterior. Esto puede reducir el tiempo de ejecución y mejorar la eficiencia del código.

-- python

```
# Antes de la optimización
for i in range(n):
    x = a * b
    y = a * b * c
    z = a * b * d

# Después de la optimización
ab = a * b
for i in range(n):
    x = ab
    y = ab * c
    z = ab * d
```

2. Eliminación de Subexpresiones Comunes: Esta técnica busca y elimina subexpresiones que se repiten en el código. Por ejemplo, si tienes una expresión como (a + b) * c + (a + b) * d, puedes calcular (a + b) una vez, almacenar el resultado en una variable temporal, y luego usar esa variable en lugar de calcular (a + b) dos veces.

-- python

```
# Antes de la optimización
result = (a + b) * c + (a + b) * d
```

```
# Después de la optimización
temp = a + b
result = temp * c + temp * d
```

3. Eliminación de Código Muerto: Se refiere a la eliminación de código que no afecta el resultado final del programa. Por ejemplo, si tienes una variable que se calcula pero nunca se usa, puedes eliminar esa variable y cualquier código que la calcule.

```
-- python
# Antes de la optimización
x = a + b
y = a - b
z = x * y
# 'x' nunca se usa después de este punto

# Después de la optimización
y = a - b
z = (a + b) * y
```

4. Transformaciones Aritméticas: Consiste en simplificar y reorganizar las operaciones aritméticas para mejorar la eficiencia. Por ejemplo, puedes reemplazar la multiplicación por una potencia de dos con un desplazamiento a la izquierda, que es una operación más rápida.

```
-- python
# Antes de la optimización
```

```
x = y * 2


# Después de la optimización
x = y << 1
```

5. Empaquetamiento de Variables Temporales: Esta técnica busca reducir el uso de memoria al reutilizar variables temporales. Por ejemplo, si tienes varias variables temporales que se usan en diferentes momentos, puedes reutilizar una sola variable temporal en lugar de tener varias.

```
-- python
# Antes de la optimización
temp1 = a + b
x = temp1 * c
temp2 = d - e
y = temp2 / f

# Después de la optimización
temp = a + b
x = temp * c
temp = d - e
y = temp / f
```

6. Mejoras en Lazos: Se refiere a la optimización de bucles en el código para mejorar su rendimiento. Por ejemplo, puedes mover cálculos que no dependen de la variable de bucle fuera del bucle, lo que se conoce como invariancia de bucle.

-- python

```
# Antes de la optimización
for i in range(n):
    x = a * b + i

# Después de la optimización
temp = a * b
for i in range(n):
    x = temp + i
```

7. Ejecución en Tiempo de Compilación: Algunas operaciones pueden ser realizadas en tiempo de compilación en lugar de en tiempo de ejecución. Por ejemplo, si tienes una expresión constante en tu código, como una operación aritmética con números literales, puedes calcular el resultado en tiempo de compilación y usar ese resultado en tu código.

-- python

```
# Antes de la optimización
x = 2 * 3 * 4 * 5

# Después de la optimización
x = 120
```

8. Eliminación de Redundancias: Esta técnica busca eliminar operaciones redundantes. Por ejemplo, si tienes una operación que se realiza varias veces con los mismos operandos, puedes realizar la operación una vez, almacenar el resultado en una variable, y usar esa variable en lugar de repetir la operación.

-- python

Antes de la optimización

x = a * b + c

y = a * b - d

Después de la optimización

temp = a * b

x = temp + c

y = temp - d

9. Cambio de Orden: Consiste en reorganizar el código para mejorar la eficiencia. Por ejemplo, puedes reorganizar las instrucciones en un bloque de código para minimizar las dependencias de datos y permitir una mayor paralelización.

-- python

Antes de la optimización

x = a + b

y = c + d

z = x * y

Después de la optimización

x = a + b

y = c + d

z = (a + b) * (c + d)

10. Reducción de Frecuencia de Ejecución (Invariancias): Se trata de mover operaciones fuera de los bucles cuando los resultados de dichas operaciones son invariantes. Por ejemplo, si tienes un cálculo que no depende de la variable de bucle, puedes mover ese cálculo fuera del bucle para que se realice una vez en lugar de en cada iteración del bucle.

-- python

```
# Antes de la optimización
for i in range(n):
    x = a * b + i

# Después de la optimización
temp = a * b
for i in range(n):
    x = temp + i
```

11. Reducción de Fuerza: Esta técnica busca reemplazar operaciones costosas con equivalentes menos costosos. Por ejemplo, puedes reemplazar una multiplicación por una adición si puedes garantizar que la multiplicación siempre se realiza por un número entero.

-- python

```
# Antes de la optimización
x = y * 2

# Después de la optimización
x = y + y
```