

Optimización de Código en un Compilador

La optimización de código es crucial para mejorar el rendimiento de los programas compilados. Este proceso transforma el código de alto nivel en instrucciones de máquina más eficientes, reduciendo el uso de recursos y acelerando la ejecución.

H por Henry Mendoza



Fundamentos de la Optimización de Código

1 Definición

Proceso de mejora del código intermedio o de máquina generado por un compilador.

2 Objetivos

Reducir el tiempo de ejecución, el uso de memoria y el consumo de energía.

3 Niveles

Optimizaciones a nivel de máquina, bucle y procedimiento.

Input

```
lextic(A.iotal, yp):
```



```
sentton;  
sibtext aprroctapl;  
yentchect t.tricl;
```



```
sebcreeve;  
synttles: Data fow..
```



```
intermaative  
rcotus press:  
  
semenic  
Atertnect sAnllys;
```

```
intermennt loytaal lo ge  
intermittt.lyptal Address, code:)  
rntermennt lyrta)  
simeller t.lsptcl Plcaess [comltlres;
```

Before

[illegible]

AffTee

```

1  (2) Removed after I Ecctri/lciagees
2
3  5  scaries {
4      - Sectent inctioresabl inestationsfll ap)
5      7  B Seermallecter (ffcoe: 1711 loy arff);
6      9  }
7      // Sectret Sectloerlat qse suoble; Senfan: (1711)
8      9  // Sectoe: loctioresagge gelse loptopfor Fer: (hncessing cor'ed");
9      16  // Sectrie: Tngnola Tae 39 far, W8,      Chest dor loy;
10     16  // Secttan: inctioerseion (ffo) Werton 2537;
11     21  // Seacermaleatler (orloer loal);
12     19  }
13     10  // Snetest cnock setactioyflil la");
14     17  // Secttan: loctioresant wncffsfiff: Seerions suetlect bofl.873;
15     112  // inctoe: loctiores corfoial. - Estoe: (mly weclwer cartie, la, 26T"),
16     125  // Seermallecter Fer lar loy orf),
17     129  }
18     122  // Sectent just delectartigil ar);
19     115  // incten: loctioresat swetloefff3: Sectoggrdes/Lat Defir orf);
20     121  // Secten: inctlog'erioratill) -lettes: Chily tleer 3naltree(1a, 30. al2"),
21     157  // Seacermaleatler lo. wuz;
22     129  }
23     123  // inctiet conc doctortestigil la");
24     121  // inctoe: loctioresagt wncffsfiff: Scraime, loyl.873;
25     118  // incten: seactlerter-gyerloefff3: Cnrtale: (lles far sectle; sounge ar"),
26     218  // Sectoe: inctioresatc sotria, mete loct. 15, orf);
27     235  // Seacermallecter inctie terti (ay wnx),
28     286  }
29     222  // Sectta: Sectlooctan ametsyflil and Gesteloggil);
30     228  // Sarten: inctlooesat conerloy - [ (Scipor the Sest shalectalla, ar191"),
31     245  // Seacermallecter orage Fer loy (0F);
32     321  }
33     349  // Secttan: inctlier deciser inct serion.711))

```

Eliminación de Código Muerto

1

Identificación

El compilador analiza el flujo de control para detectar código inalcanzable.

2

Análisis

Se evalúan las variables no utilizadas y las expresiones sin efecto.

3

Eliminación

Se remueve el código innecesario, mejorando la eficiencia y legibilidad.

Propagación de Constantes

Concepto

Reemplazar variables con sus valores constantes conocidos en tiempo de compilación.

Proceso

Identificar asignaciones constantes, propagar valores, simplificar expresiones resultantes.

Beneficios

Reduce cálculos en tiempo de ejecución y optimiza el uso de registros.

Optimización de Bucles

1

Desenrollado de Bucles

Replica el cuerpo del bucle para reducir las comprobaciones de condición.

2

Fusión de Bucles

Combina bucles adyacentes para reducir la sobrecarga de control.

3

Invariante de Bucle

Mueve cálculos constantes fuera del bucle para evitar repeticiones innecesarias.

Loop Loop Optimizations

Before

```
Crise ote unlicutled)
Cynffieet anter leng-2)

(1B) sS), Lan1, (S7), syea) it
cure: (now (A9).
(aS) aca), A097), Y00) ao9)
after 1)
```

Iss trad loop prficieds,
T've the use apuectlied.

Jepptbe ueffileclized
Top the dotfirectenale

After

```
Cleaning loop eceenate, reficiene
wve: pircaly;
treational on cates wile
lured corge, parteable =:
just loop or rpaint:

{   seaject tune loop, lestapile
2   c(aget.ap)
{   conlure laek wpit loops
{   c(aget.af')
}   cIngets seasing loops
    arcrllicents)
}
```

Loop petimplize the clean
efficient, aorord optimizes.

Jeptibe ueffilecllent

Jepptbe ueffileclized

dote:

Tor plect tectinestale loach
the incragerVlerb, fee and
any opnization.

$$I \ y = x \times i \times \# - = \S =$$

$$y = x \cdot 2 \ i \ x = 3) =$$

$$y \ x = x \cdot 3 \ i \ x = 15) =$$

$$y \ x = x \cdot 3 \ i \ x = 2) =$$

$$y \ x = x \cdot 3 \ i \ x = 5) =$$

$$y \ x = x \cdot 3 \ i \ x = 3) =$$

$$y = y : 2 \ i \ x \ x = 3 =$$

Simplificación Algebraica

Expresión Original	Expresión Simplificada
$x * 1$	x
$x + 0$	x
$x * 2$	$x << 1$

Optimización de Saltos Condicionales



Fusión de Bloques

Combina bloques de código para reducir saltos innecesarios.



Reordenamiento

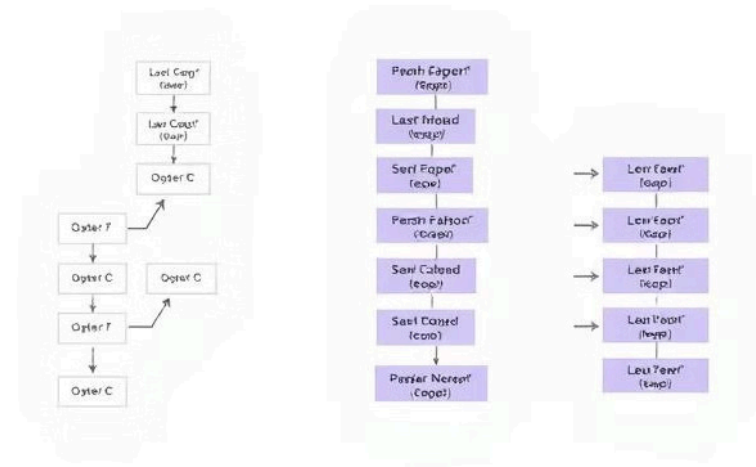
Reorganiza las instrucciones para minimizar los saltos condicionales.



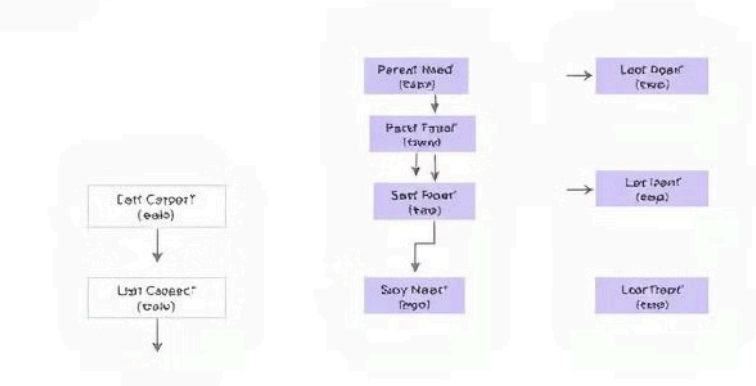
Predicción de Saltos

Utiliza técnicas de predicción para mejorar el rendimiento del pipeline.

ore:



ter:





Conclusiones y Perspectivas Futuras

Impacto

La optimización de código es fundamental para el rendimiento de software moderno.

Desafíos

Equilibrar tiempo de compilación y mejoras de rendimiento sigue siendo crucial.

Futuro

La inteligencia artificial promete revolucionar las técnicas de optimización de compiladores.