Compilerbouw
Peephole optimizer

Alexandra Moraga Pizarro (6129544)
Tamara Ockhuijsen (6060374)
Fredo Tan (6132421)

31 december 2011

1 Inleiding

Dit verslag beschrijft de uitwerking van de praktische opdracht voor het vak Compilerbouw, gegeven aan de Universiteit van Amsterdam in het jaar 2011/2012. Voor deze opdracht is een peephole optimizer voor SimpleScalar DLX assembly code ontwikkeld. Er zijn verschillende benchmark programma's in de taal C meegegeven. Een gcc cross compiler kan deze C-code compileren in assembly code die dient als invoer voor de peephole optimizer. Deze leest de assembly code en genereert nieuwe assembly code waarin overbodige instructies zijn verwijderd. De functionaliteit blijft hetzelfde als die van de originele assembly code, maar het programma werkt wel sneller.

2 Implementatie

De code kan gerund worden op de volgende manier: `python main.py benchmarks/assemblyfile.s`

De PLY package is gebruikt voor het parsen van assembly code. Dit is een ingebouwde implementatie van lex en yacc voor Python.

De implementatie is onderverdeeld in het parsen met lex en yacc en de optimalisatie.

2.1 Lex en yacc

Door gebruik te maken van de PLY (Python Lex-Yacc) package was het mogelijk om een parser voor SimpleScalar assembly code te schrijven.

PLY bestaat uit twee afzonderlijke modules: lex.py en yacc.py, die beide te vinden zijn in een pakket met de naam python-ply. De lex.py module wordt gebruikt om invoertekst op te breken in gedefinieerde tokens door middel van een verzameling van reguliere expressieregels. yacc.py wordt gebruikt om de syntaxis te herkennen die is opgegeven in de vorm van een context vrije grammatica. yacc.py maakt gebruik van LR parsing en genereert de parsing tabellen met behulp van de LALR (1) tabelgenerator algoritme.

De twee tools zijn bedoeld om met elkaar te werken. Concreet zorgt lex.py voor een externe interface in de vorm van een `token()` functie die de volgende geldige token van de invoer retourneert. yacc.py haalt hier herhaaldelijk tokens vandaan en roept grammaticaregels op.

De lex en yacc implementaties zijn te vinden in `parse_lex.py` en `parse_yacc.py`

2.2 Optimalisatie

In `block_optimize.py` wordt er gezocht naar basic blocks door middel van een expressielijst, hier worden vervolgens de hoofdexpressies uitgekozen door naar de jump targets te kijken en deze vervolgens op te slaan.

`peep.py` leest expressies en voegt expressies toe. Daarnaast filtert en vervangt hij deze expressies. Vervolgens wordt er bepaald of het geldige expressies zijn. Tenslotte wordt er een assembly code aangemaakt door middel van een expressielijst en de code wordt opgeslagen in de daarvoor bestemde file.

`optimize.py` kijkt of er overbodige instructies staan in de assembly code die verwijderd of vereenvoudigd kunnen worden. De volgende instructies worden geoptimaliseerd in de assembly code.

```
Original sequence        Replacement

mov $regA,$regB          ---

mov $regA,$regB
instr $regA, $regA,...   instr $regA, $regB,...

instr $regA,...          instr $4,...
mov $4, $regA            jal XXX
jal  XXX

sw $regA,XXX             sw $regA, XXX
ld $regA,XXX

shift $regA,$regA,0      ---

add $regA,$regA,X        lw ...,X($regA)
lw ...,0($regA)

  beq ...,$Lx              bne ...,$Ly
  j $Ly                  $Lx:
$Lx: ...
```

3 Resultaten

Voor elk C programma is het aantal instructies aan het begin, na de branch optimalisaties en na de basic blocks optimalisatie berekend. De resultaten staan in onderstaande tabel. In de laatste kolom is het percentage van de mate van optimalisatie te vinden.

Programma	Begin	Geoptimaliseerd	Basic Blocks	Percentage
acron.s	432	414	407	3,9%
clinpack.s	3728	3689	3681	1,2%
dhrystone.s	892	885	876	1,8%
pi.s	121	120	120	0,8%
slalom.s	4610	4551	4548	1,3%
whet.s	1032	1021	1019	1,2%

4 Conclusie

Met behulp van de optimalisaties van de globale branch instructions en optimalisaties op de basic blocks is het gelukt om het aantal instructies van assembly code te reduceren.

Hoewel het optimaliseren wel gelukt is, blijven de percentages minimaal. De percentages waren hoger geweest als de advanced optimalisaties waren geïmplementeerd.

5 Referenties

The SimpleScalar Tool Set, Version 2.0
`http://www.science.uva.nl/~andy/compiler/users_guide_v2.pdf`
Python Lex-Yacc
`http://www.dabeaz.com/ply/`
Practicumopdracht
`http://staff.science.uva.nl/~andy/compiler/prac.html`