Optimizing Compilers
2nd Lecture

Bernhard Scholz
Institut f. Computersprachen
Argentinierstr. 8
scholz@complang.tuwien.ac.at

Overview

- Jump Optimization
- Basic Block Straightening
- Basic Block Reordering
- MiniC Framework
- Demo

Jump Optimization

- Problem
  Intermediate code generator produces
  - jumps to jumps
  - jumps to conditional jumps
  - useless jumps
  - conditional jumps to jumps
- Effect
  - reduce code size
  - improve run-time
- Approach
  - search jump-templates
  - code replacement

Jump Elimination

- Jumps to jumps

```
 goto L1;
 ....
L1: goto L2;
```

```
 goto L2;
 ....
(L1: goto L2;)
```

If "**L1:goto L2;**" is not further used , remove it!

- Conditinal jumps to jumps

```
 if (b) goto L1;
 ....
L1: goto L2;
```

```
if (b) goto L2;
 ....
(L1: goto L2;)
```

If "**L1:goto L2;**" is not further used , remove it!

Example for Jump Elimination

```
loop:
 b = i < 10;
 if (b) goto l1;
 goto exit;
l1:
 goto l2;
exit:
 goto not_here;
l2:
 i:=i+1;
 goto loop;
not_here:
 …
```

```
loop:
 b = i < 10;
 if (b) goto l2;
 goto not_here;



l2:
 i:=i+1;
 goto loop;
not_here:
 …
```

Swapped condition: one "goto l2" can be eliminated!

Jump Elimination(2)

- Useless Jumps

```
  ...
 goto L1;
L1:
```

```
 ...
(L1:)
```

"**L1:**" might be still a target. If not, remove it!

- Jumps to conditional jumps

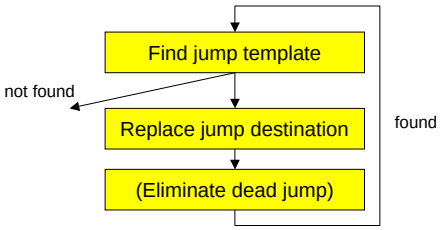
```
 goto L1;
  ...
L1:if (b)goto L2;
L3:
```

```
 if (b) goto L2;
goto L3;
 ...
(L1:if (b)goto L2;)
L3:
```

"**L1:**" might be still a target. If not, remove it!

Approach for Jump Elimination

- Approach
  - iterative recognition and replacement of jumps
  - e.g. there can be jumps to jumps to jumps …..



Example for Jump Elimination(2)

- Jumps to Jumps:

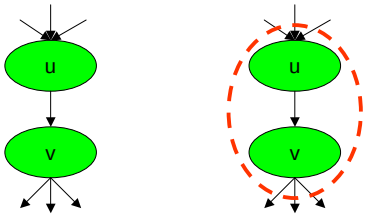
```
 goto l1;
l1:
 goto l2;
l2:
 goto l3;
l3:
 goto l4;
l4:
```

```
 goto l4;
l1:
 goto l4;
l2:
 goto l4;
l3:
 goto l4;
l4:
```


Basic Block Straightening

- Problem
  - connect two consecutive blocks
- Effect
  - reduce code size (eliminate jumps)
  - improve run-time
- Approach
  - check control-flow graph for consecutive blocks
  - aggregate blocks

Basic Block Straightening(2)



- Analysis: find an edge $(u,v) \in E$ such that $succs(u)=\{v\} \wedge preds(v)=\{u\}$.
- Transformation: merge basic blocks and remove jump
- Iterative approach: until no basic block can be merged

Example

```
 a=10;
 goto l2;
l1:
 c=a*b;
 goto l3;
l2:
 b=a+2;
 goto l1;
 …
l3:
```

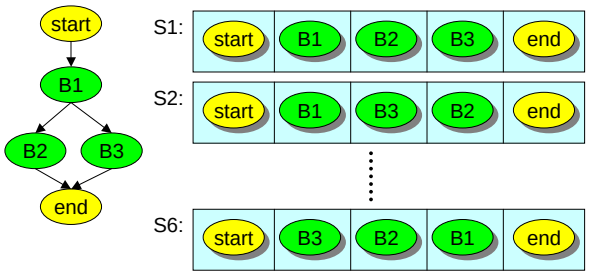
```
 a=10;
 b=a+2;
 c=a*b;
 goto l3;
 ...
l3:
```

- Analysis: find an edge $(u,v) \in E$ such that $succs(u)=\{v\} \wedge preds(v)=\{u\}$.
- Transformation: merge basic blocks and remove jump

Basic Block Reordering

- Problem
  - How can I order basic blocks such that the number of jumps get minimal?
  - Generalized approach for basic block straightening
- Effect of Basic Block Reodering
  - reduce code size (eliminate jumps)
  - improve run-time
- Approach
  - Solve it and get the chocolate award!

Sequences of Basic Blocks



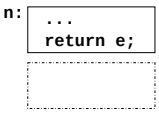
- Different sequences of basic blocks yield different costs
- There are $N!$ different sequences
- Obvious: brute force is too expensive.

Costs of a Sequence

- Local costs
  - jumps at the end of basic block cause costs
  - at most two jumps at the end of a basic block
  - suppose that the branch frequency is given, i.e $f: \mathrm{E} \to \mathrm{Z}_+$
  - assume that there are constant costs for a jump $j: \mathrm{E} \to \mathrm{Z}_+$
  - local costs constitutes of the decision which basic block is the subsequent basic block
  - introduce sequence predicate $\mathrm{seq}(n_1,n_2)$ for two basic blocks
  - define local cost function depending on seq. predicate $cost: \mathrm{N} \to \mathrm{Z}_+$
  - there are only three cases of basic blocks

Case 1

- There is no successor of node n
- Successor is arbitrary



- Cost Function

$$\mathrm{cost}(n) = 0$$

Case 2

- One successor node s of node n

$\neg$seq(n,s):

```
n:  ...
    goto l1;

s:  l1:
    ...
```

seq(n,s)

```
n:  ...

s:  ...
```

- Cost Function

$$\text{cost}(n) = \begin{cases} 0, & \text{seq}(n,s) \\ j(n,s)\,f(n,s), & \text{otherwise} \end{cases}$$

Case 3

- Two successors s_1 and s_2 of node n

$\neg$seq(n,s_1) $\wedge\neg$seq(n,s_2)

```
n:  ...
    if(b) l2;
    goto l1;

s1: l1:
    ...

s2: l2:
    ...
```

seq(n,s_1) $\wedge\neg$seq(n,s_2)

```
n:  ...
    if(b) l2;

s1: l1:
    ...

s2: l2:
    ...
```

$\neg$seq(n,s_1) $\wedge$seq(n,s_2)

```
n:  ...
    if(!b) l1;

s2: l2:
    ...

s1: l1:
    ...
```

$$\text{cost}(n) = \begin{cases} j(n,s_1)\,f(n,s_1) + j(n,s_2)\,f(n,s_2), & \neg\text{seq}(n,s_1) \wedge \neg\text{seq}(n,s_2) \\ j(n,s_2)\,f(n,s_2), & \text{seq}(n,s_1) \wedge \neg\text{seq}(n,s_2) \\ j(n,s_1)\,f(n,s_1), & \text{otherwise} \end{cases}$$

Optimization Problem

- Global costs should be minimal

$$\sum_{n \in N} \text{cost}(n) \rightarrow \text{minimal}$$

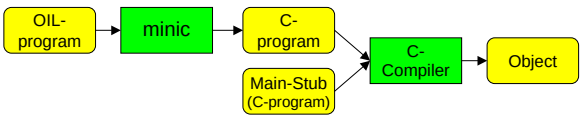
- Correctness issue
  - only one node can be a subsequent node of another one (except start node)
- Algorithm???

  Try hard and get the chocolate award!!

miniC Project

- OIL language features:
  - intermediate language
  - quad-code
  - subset of C without types
  - restricted control statements (if, goto, labels)
  - no global variables, simple functions
- compiler already implemented (source-to-source)
- add optimizations (analysis/transformations)

Translation



- MiniC compiler does not provide main() function
- write your own main-stub that calls OIL functions
- compile MiniC to C
- compile main-stub
- link generated C-code and main-stub
- use Makefiles to drive the whole code generation

miniC Example

OIL

```
// factorial number
fac(x)
{
 b = x <= 1;
 if (b) goto L1;
 a = x - 1;
 r = fac(a);
 r = x * r;
 return r;
L1:
 return 1;
}
```

C-Output

```
long fac(long __x)
{
 long __r,__a,__b;

 (__b = (__x <= 1));
 if (__b) goto _L1;
 (__a = (__x - 1));
 (__r = fac (a));
 (__r = (__x * __r));
 return __r;
_L1: ;
 return 1;
}
```

miniC Framework

- basic functionality is implemented
- available as tarball
  www.complang.tuwien.ac.at/scholz/lecture/
- not mandatory: you can develop from scratch
- untar: **`tar zxvf minic.tgz`**
- build miniC-compiler (in src/ directory): **`make`**
- compile miniC-testcode (in example/ directory):
  `./minic –otest.c test.mc`
- compile produced C-Code in (example/ directory):
  `gcc test.c test_main.c`
- execute (in example/ directory):
  `./a.out`

miniC Source Files

- source directory src/

Module	Description
scan.l	scanner
parser.y	syntax analysis, build internal data structure
output.c	functions to produce C-code
symtab.c	build symbol table
alloc.c	helper functions to allocate data structures
minic.h	header file
Makefile	makefile to build minic
sbitmap.[c|h]	set library

- example directory example/
  - bubble.mc, fac.mc, and main-stubs
- script ./run for building framework and examples

miniC Data Structures

- keep it as simple as possible
- efficiency is not (so) important
- data structures
  - expressions (struct expr)
  - statements (struct stmt)
  - functions (struct func)
  - symbol lists (struct sym)
- simply linked lists
- don‘t care about memory leaks

Stop

- Next lecture: 3.4.2002, 13:45 – 14:45
- No lecture next week!!
- Hand in 1st assignment!

This document was created with Win2PDF available at http://www.daneprairie.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.