编译原理/Compiler
Basic Blocks and Traces

邱锡鹏 周雅倩
复旦大学计算机科学技术学院
{xpqiu, zhouyaqian}@fudan.edu.cn
2018/11/23

References

- http://www.stanford.edu/class/cs143/
  - Partial contents are copied from the slides of Alex Aiken

Tree language V.S. machine languages

- There are certain aspects of the tree language that do not correspond exactly with machine language.
  - The Cjump
  - ESeq
  - Call
- Some aspects of the Tree language interfere with compile-time optimization analysis.

Issues

- IR branching does not translate well into Assembly branching

```
if CONDITION then
 BLOCK 1
else
 BLOCK 2
```
High-level Language

```
BNE $r1, $r2, label_1
BLOCK 2
label_1: BLOCK MIPS64
```
Assembly

CANONICAL TREE

Canonical Tree

- For the purposes of code optimization, the compiler should be able to evaluate parts of the IR tree in any order it sees fit.
- Issues:
  - subexpressions (parts of the IR tree) may have side effects
  - different order of evaluating functions can have different end effects
- Canonical Tree
  - tree where any subtree can be evaluated in any order (informal definition)

Canonical Tree

- **Definition**:
  - No Seq or ESeq.
  - The parent of each Call is either EXP(…) or MOVE(Temp t,…).
- **Linearize**
  - removes the ESeqs and
  - moves the Calls to top level

Rewrite Tree

- Rewrite IR tree to canonical tree
- Construst basic block
- Find traces and adjust Cjump

Canonical Tree

- ESeq(s1, ESeq(s2, e))=> ESeq(Seq(s1,s2), e)
- BinOP(op, Eseq(s, e1), e2) =>
  ESeq(s, BinOP(op, e1, e2))
- Mem(ESeq(s,e1)) => ESeq(s, Mem(e1))
- Jump(ESeq(s, e1)) => Seq(s, Jump(e))
- Cjump(op, ESeq(s, e1), e2, l1,l2) =>
  Seq(s, Cjump(op, e1, e2, l1,l2)

Canonical Tree

- BinOP(op, e1, ESeq(s, e2)) =>
  ESeq(MOVE(Temp t, e1),
  ESeq(s, BinOP(op, Temp t, e2)))
- Cjump(op, e1, ESeq(s, e2), l1, l2) =>
  Seq(MOVE(Temp t, e1),
  Seq(s, Cjump(op, Temp t, e2, l1,l2)))

Canonical Tree

- If s, e1 commute
  - BinOP(op, e1, ESeq(s, e2)) =>
    ESeq(s, BinOP(op, e1, e2))
  - Cjump(op, e1, ESeq(s, e2), l1,l2) =>
    Seq(s, Cjump(op, e1,e2, l1,l2)

Moving calls to top level

- BinOP(Plus, Call(…), Call(…))
- Call(fun, args) =>
  ESeq(MOVE(Temp t, Call(fun, args)),
  Temp t)

Linearize

- Seq(Seq(a,b),c) = Seq(a,Seq(b,c))
- Seq(s_1,Seq(s_2,…,Seq(s_{n-1},s_n)…) => s_1,s_2,…,s_{n-1},s_n
  - si do not include Seq or Eseq.

BASIC BLOCKS AND TRACES

Taming Conditional Branches

- Two-way branch has no counterpart in most machines
- Rearrange the trees so that every Cjump(cond, lt,lf) is immediately followed by Label(lf)
- Two stages
  - Takes a list of canonical trees form them into basic blocks
  - Order the basic blocks into a trace

Basic Block

- A basic block is
  - a sequence of statements
  - that is always entered at the beginning and
  - exited at the end
- That is
  - The first statement is a Label
  - The last statement is a Jump or Cjump
  - There are no other Labels, Jumps or Cjumps

Basic Block

- Scan from beginning to end
  - A Label is found, a new block is started (and previous block is ended)
  - A Jump or Cjump is found, a block is ended (and the next block is started)
  - If there is a block not ending with a Jump or Cjump, then add a Jump to next block's label
  - If there is a block not begin with a Label, then add a new label at the beginning of the block

Trace

- Basic block can be arranged in any order
- Choose an ordering of the blocks satisfying the condition that Cjump is followed by its false label
- Try to let Jumps are immediately followed by their target label
  - May delete the jump

Trace

- A trace is a sequence of statements
  - that could be consecutively executed
  - during the execution of the program
- We want to make a set of traces that exactly covers the program

Trace

Put all the blocks of the program into a list Q
while Q is not empty
  start a new (empty) trace, call it T
  remove the head element b from Q
  while b is not marked
    mark b; append b to the end of the current trace T;
    examine the successors of b
    if there is any unmarked successor c
      b ← c

Finishing Up

- For any Cjump followed by its false label
  - we let it alone
- For any Cjump followed by its true label
  - Switch the true and false labels and
  - Negate the condition
- For any Cjump(cond, a, b, l_t, l_f) followed by neither label
  - Cjump(cond, a, b, l_t, l'_f)
  - Label l'_f
  - Jump(NAME l_f)

Optimization