Dynamic Compilation

do the dynamic compile once we know values of variables that will remain invariant thruout execution but unknown until then

once: generate code that has been optimized; applying any number of traditional

than optimize for the more general case

to give intuition of why we generate better code

determines loop limit -->

- loop induction variable & any variables derived from it = RTC

path not taken

whole point is performance

Our Dynamic Compiler Framework

short form first

static compiler:

- good quality static compiler (Multiflow)
- generates machine code templates that assume they have been done; holes

dynamic compiler

- hardly a compiler, template manipulator

design analyses to support full functionality of C

Dynamic Compilation System

see what opts we apply & effect of applying them

before how the framework applies them

capability today

Cache Lookup Example

written as general program, able to handle

specific config at run-time: cache parameters = RTC

explain what it does

- internally

Cache Lookup Example

red code calculates RTC & does opts based on RTC

output of dynamic compiler framework: specialized for 4-way... (machine code)

our static analysis recognizes RTC; derive additional RTC from them: calculation whose operands ...

perform opts on them

variable reachable from a RTC = RTC

mult 2 RTCs = RTC; by constant folding obtain value

determine if power of 2; yes ==> replace expensive div by cheap right shift

RTC controls loop termination; complete LU

offset is induction variable; different values in each iteration

recap what opts have been done; result is half # instructions; half # loads

Dynamic Compiler Framework

main parts

annotations

- for partially constant data structure, **dynamic** on memory dereference operators
- multiple compiled versions for a single DR, each optimized for diff RTCs
- balance between manual (‘C) & automatic (no one)

static code

template

set-up

directives

set-up code executed once

Dynamic Compiler Framework

black code

Dynamic Compiler Framework

red code

calculates values of induction variable that are used as offset into array in each iteration

Dynamic Compiler Framework

table of RTC values; loops have a nested structure; follow the structure guided by directives

stitcher puts executable template into malloc'd space

- address of space in global var; indirect jump thru it
- another indirect jump out of it

summarize diagram

executed once

Dynamic Compiler Framework

Run-time constants identification

- computes variables that are constant at each program point
- start with annotated set: add to it
- rich set of data structures (arrays, pointer-based structures)

Reachability analysis

- refines; handles case of RTC that lie on paths after a condition that is RTC
- after merge, var = RTC only if has same constant reaching def (same SSA subscript) on both paths
- useful for `if` with RTC predicate; only one path will be taken; constants on it
- determines restricted conditions (expressed as branch outcomes) under which program points can be reached after dynamic compilation
- at merge: reachability analysis determines if conditions are mutually exclusive (union)
- loop head merges have mutually exclusive conditions
- rich set of control structures (`if`, `switch`, unstructured control flow)

use both sets of RTCs & reachability conditions for previous program point to calculate for current program point

RA & CS for DR and surrounding code

- don't move holes outside template boundaries
- no copy prop of holes outside DR
- induction variables in unrolled loop are not loop invariant, so can't hoist: each iteration has own copy

Dynamic Compiler Framework

traditionally

fast DC code --> needed an optimizing DC

fast dynamic compiler --> pay in efficiency of code generated

strive for best of both worlds

Current Applications

Current application suite

Certain point in our work:

- speedups
  - what opts pay off
  - how widely can we apply them
  - what new opts do we need
- debugging, some are algorithms
- results for 2: used in other EKH96
- matrix
  - 3000x3000 matrix, scaling factor 1-100, take the average
  - strength reduction
- calculator
  - polynomial in x & y, 30 pairs of values, take the average
  - all others

Future Directions

each student takes an application area

Related Work

PE

- phased compilation
- BTA + static analyses, specialization done in stitcher
- source-to-source, analysis on higher level ASTs
- purely functional languages

Related Work

Fabius

- PE framework
- functional lang: no side effects (RTCs don't change)
- currying triggers DC: for each argument, new function is compiled
  BTA identifies calculations that depend on first argument
- safe; our correctness depends on correct programmer annotations

Related Work

Consel & Noel

- pointer analysis unsafe with casts & pointer arithmetic

Related Work

‘C

- DCG: library for constructing expression trees

Related Work

Guenter: static special casing; jump to the case

- if DC, can do better: specialize code for value, not load from table