Překladače (8. přednáška)

Jan Hubička

Katedra aplikované matematiky
Univerzita Karlova
Praha

4. května 2020

Dominance

Definice (Prosser 1959)

Basic blok *a* dominuje basic blok *b* pokud na každé cestě z Entry do *b* je *a*.

Pozorování (tranzitivita)

Pokud *a* dominuje *b* and *b* dominuje *c* potom *a* dominuje *c*.

Důsledek: dominance je částečné uspořádání.

Pozorování

Pokud *a* i *b* dominují *c*, potom *a* dominuje *b* nebo *b* dominuje *a*

Důsledek: dominance tvoří strom zakořeněný v Entry.
Důsledek: dominance má kompaktní stromovou reprezentaci pomocí preorderu a postorderu.

Single static assignment

Definitice (SSA)

Program je v SSA formě pokud každá proměnná je přiřazena právě jednou.

Pokud nechceme používat nedefinované proměnné, každá definice musí dominovat použití.

Oprace ϕ

Operace ϕ se používá jen na začátku basic bloku. Má tolik parametrů kolik je vstupních hran CFG. Hodnota operace ϕ je parametr odpovídající hraně po které program do basic bloku přišel.

SCCP: Sparse conditional constant propagation: Wegman, Zadeck, 1991

Propagační engine

```
SSAwl $\leftarrow \emptyset$, CFGwl $\leftarrow \emptyset$.
Pro každý basic blok $B$: exec($B$) $=$ false
Dokud SSAwl $\cup$ CFGwl $\neq \emptyset$
    Dokud CFGwl $\neq \emptyset$
        Vyzvedni $B$ z CFGwl, exec($B$) $\leftarrow$ true
        Vyhodnoť každé $\phi$ a každý výraz v $B$
        Pokud $B$ ma jen jednoho potomka, přidej ho do CFGwl.
    Dokud SSAwl $\neq \emptyset$
        Vyzvedni $s$ z SSA$wl$ a vyhodnoť definici $s$ do $t$.
        Pokud $t \neq$ value($n$)
            Přidej do SSAwl všechny použití $p$ hodnoty $s$ takové, že exec(bb($p$)) $=$ true
```

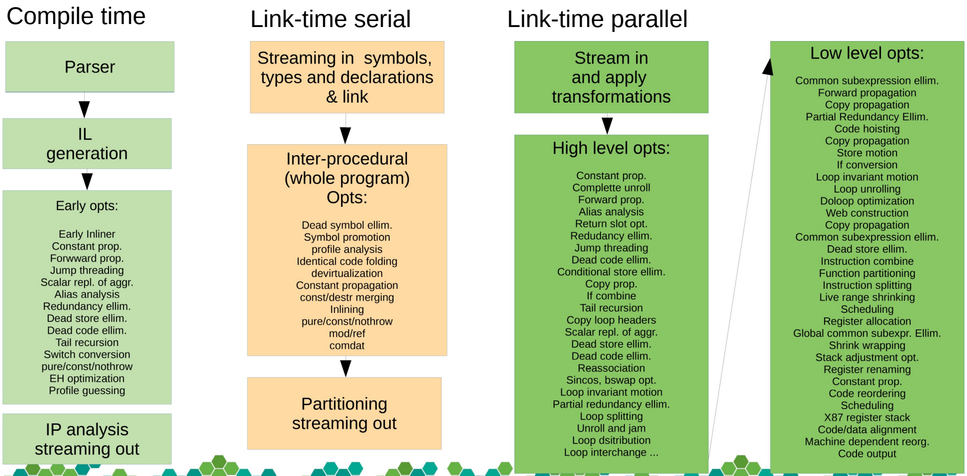
EvalPhi(s)

Uvažuje jen ty parametry jejiž bloky mají nastavený exec.

EvalStmt(s)

Po vyhodnocení podmínky na známou konstatu nebo $\perp$ přidá odpovídající bloky do CFGwl.

GCC optimization pipeline



Compile time

Parser

IL generation

Early opts:

Early Inliner
Constant prop.
Forwward prop.
Jump threading
Scalar repl. of aggr.
Alias analysis
Redundancy ellim.
Dead store ellim.
Dead code ellim.
Tail recursion
Switch conversion
pure/const/nothrow
EH optimization
Profile guessing

IP analysis streaming out

Link-time serial

Streaming in symbols, types and declarations & link

Inter-procedural (whole program) Opts:

Dead symbol ellim.
Symbol promotion
profile analysis
Identical code folding
devirtualization
Constant propagation
const/destr merging
Inlining
pure/const/nothrow
mod/ref
comdat

Partitioning streaming out

Link-time parallel

Stream in and apply transformations

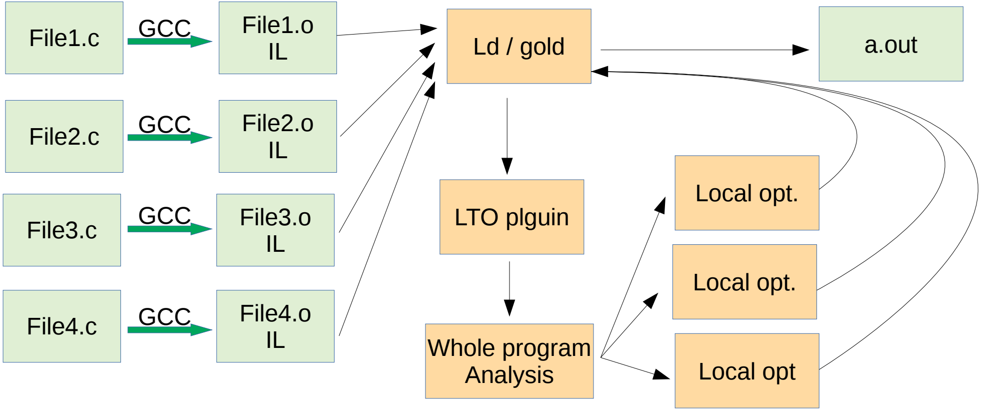
High level opts:

Constant prop.
Complette unroll
Forward prop.
Alias analysis
Return slot opt.
Redudancy ellim.
Jump threading
Dead code ellim.
Conditional store ellim.
Copy prop.
If combine
Tail recursion
Copy loop headers
Scalar repl. of aggr.
Dead store ellim.
Dead code ellim.
Reassociation
Sincos, bswap opt.
Loop invariant motion
Partial redundancy ellim.
Loop splitting
Unroll and jam
Loop dsitribution
Loop interchange ...

Low level opts:

Common subexpression ellim.
Forward propagation
Copy propagation
Partial Redundancy Ellim.
Code hoisting
Copy propagation
Store motion
If conversion
Loop invariant motion
Loop unrolling
Doloop optimization
Web construction
Copy propagation
Common subexpression ellim.
Dead store ellim.
Instruction combine
Function partitioning
Instruction splitting
Live range shrinking
Scheduling
Register allocation
Global common subexpr. Ellim.
Shrink wrapping
Stack adjustment opt.
Register renaming
Constant prop.
Code reordering
Scheduling
X87 register stack
Code/data alignment
Machine dependent reorg.
Code output

Parallelized Link-time optimization (WHOPR)



DCE (Dead Code Ellimination)

DCE nad SSA: jednoduchý mark and sweep algoritmus.

DCE (Dead Code Ellimination)

DCE nad SSA: jednoduchý mark and sweep algoritmus.
Jak ale mazat nedůležité control flow?

DCE (Dead Code Ellimination)

DCE nad SSA: jednoduchý mark and sweep algoritmus.
Jak ale mazat nedůležité control flow?

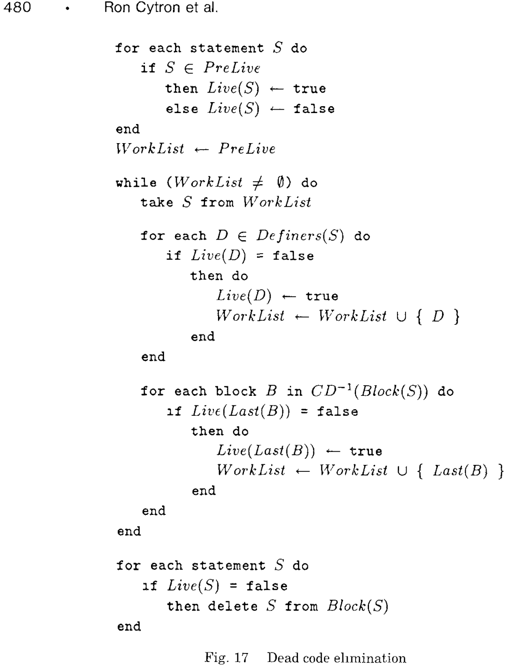
Postdominance

Basic blok *a* postdominuje basic blok *b* pokud na každé cestě *b* do Exit je *a*.

Control dependence

Basic block *a* má závislost (control dependence) na *b* pokud platí následující podínky

1. Existuje cesta z *b* do *a* taková, že *a* posdominuje každý blok za *b*.
2. *a* není striktně posdominovaný *b*.



```
for each statement $S$ do
   if $S \in PreLive$
      then $Live(S)$ ← true
      else $Live(S)$ ← false
end
$WorkList \leftarrow PreLive$

while ($WorkList \neq \emptyset$) do
   take $S$ from $WorkList$

   for each $D \in Definers(S)$ do
      if $Live(D)$ = false
         then do
            $Live(D)$ ← true
            $WorkList \leftarrow WorkList \cup \{ D \}$
         end
   end

   for each block $B$ in $CD^{-1}(Block(S))$ do
      if $Live(Last(B))$ = false
         then do
            $Live(Last(B))$ ← true
            $WorkList \leftarrow WorkList \cup \{ Last(B) \}$
         end
   end
end

for each statement $S$ do
   if $Live(S)$ = false
      then delete $S$ from $Block(S)$
end
```

Fig. 17 Dead code elimination

Value numbering

Value numbering je analýza, která rozdělí výrazy do tříd ekvivalence. Dva výrazy je vstejné třídě ekvivalence mají vždy stejnou hodnotu.