

GVN-Hoist: Hoisting Computations from Branches

Sebastian Pop and Aditya Kumar

SARC: Samsung Austin R&D Center

November 3, 2016

CFGSimplify's code hoisting

- ▶ hoists computations at the beginning of BB
- ▶ uses operands equality to detect same computations
- ▶ stops at first difference
- ▶ very fast: disabling it slows the compiler: 1688 \rightarrow 1692 Bn insns (callgrind compiling the test-suite on x86_64-linux)

CFGSimplify limits

Original program

```
inv = 1/d;  
if (inv >= 0) {  
    ta = a * inv;  
    tb = b * inv;  
} else {  
    ta = b * inv;  
    tb = a * inv;  
}
```

CFGSimplify limits

Original program

```
inv = 1/d;  
if (inv >= 0) {  
    ta = a * inv;  
    tb = b * inv;  
} else {  
    ta = b * inv;  
    tb = a * inv;  
}
```

→

Expressions hoisted

```
inv = 1/d;  
x = a * inv;  
y = b * inv;  
if (inv >= 0) {  
    ta = x;  
    tb = y;  
} else {  
    ta = y;  
    tb = x;  
}
```

GVN-Hoist: Hoisting Computations from Branches

- ▶ removes all limitations of CFGSimplify implementation
- ▶ works across several BBs: hoists to a common dominator
- ▶ hoist past ld/st side effects: uses Memory-SSA for fast dependence analysis
- ▶ reduces code size
- ▶ reduces critical path length by exposing more ILP

Optimistic GVN-hoist Algorithm

1. compute value number of scalars, loads, stores, calls
2. compute insertion points of each type of instructions
3. hoist expressions and propagate changes by updating SSA

GVN: Value Numbering Example and Limitations

Simple program

```
a = x + y
b = x + 1
c = y + 1
d = b + c
e = a + 2
f = load d
g = load e
```

GVN: Value Numbering Example and Limitations

Simple program

```
a = x + y
b = x + 1
c = y + 1
d = b + c
e = a + 2
f = load d
g = load e
```

→

Value Numbering

```
(a, 1)
(b, 2)
(c, 3)
(d, 4)
(e, 4)
```


GVN: Value Numbering Example and Limitations

Simple program

```
a = x + y
b = x + 1
c = y + 1
d = b + c
e = a + 2
f = load d
g = load e
```

→

Value Numbering

```
(a, 1)
(b, 2)
(c, 3)
(d, 4)
(e, 4)
```

Limitations to current GVN implementation

```
(f, 5)
(g, 6)
// should be (g, 5)
```

GVN-Hoist Step 1: Collect Value Numbers

- ▶ scalars: use the existing GVN infrastructure

current GVN not accurate for loads and stores: use ad-hoc change

- ▶ loads: VN the gep
- ▶ stores: VN the gep and stored value
- ▶ calls: as stores, loads, or scalars (following calls' side-effects)

GVN-Hoist Step 2: Compute Insertion Points

insertion point: location where all the operands are available

- ▶ compute a common insertion point for a set of instructions having the same GVN (similar to VBEs but not as strict)
- ▶ partition the candidates into a smaller set of hoistable candidates when no common insertion points can be found

GVN-Hoist Step 3: Move the Code

- ▶ scalars: just move one of the instructions to the hoisting point and remove others; update SSA
- ▶ loads and stores: make geps available, then hoist; update SSA and Memory-SSA

Example

```
define float @f(float %d, float %min, float %max, float %a) {
entry:
    %div = fdiv float 1.000000e+00, %d
    %cmp = fcmp oge float %div, 0.000000e+00
    br i1 %cmp, label %if.then, label %if.else

if.then:
; preds = %entry
    %sub = fsub float %min, %a
    %mul = fmul float %sub, %div
    %sub1 = fsub float %max, %a
    %mul2 = fmul float %sub1, %div
    br label %if.end

if.else:
; preds = %entry
    %sub3 = fsub float %max, %a
    %mul4 = fmul float %sub3, %div
    %sub5 = fsub float %min, %a
    %mul6 = fmul float %sub5, %div
    br label %if.end

if.end:
; preds = %if.else, %if.then
    %tmax.0 = phi float [ %mul2, %if.then ], [ %mul6, %if.else ]
    %tmin.0 = phi float [ %mul, %if.then ], [ %mul4, %if.else ]
    %add = fadd float %tmax.0, %tmin.0
    ret float %add
}
```

Example

```
define float @f(float %d, float %min, float %max, float %a) {
entry:
    %div = fdiv float 1.000000e+00, %d
    %cmp = fcmp oge float %div, 0.000000e+00
    %sub1 = fsub float %max, %a
    %sub = fsub float %min, %a
    %mul2 = fmul float %sub1, %div
    %mul = fmul float %sub, %div
    br i1 %cmp, label %if.then, label %if.else

if.then:
; preds = %entry
    br label %if.end

if.else:
; preds = %entry
    br label %if.end

if.end:
; preds = %if.else, %if.then
    %tmax.0 = phi float [ %mul2, %if.then ], [ %mul, %if.else ]
    %tmin.0 = phi float [ %mul, %if.then ], [ %mul2, %if.else ]
    %add = fadd float %tmax.0, %tmin.0
    ret float %add
}
```

Cost models

tuned on x86_64 and AArch64 Linux: test-suite, SPEC 2k, 2k6, ...

- ▶ limit the number of basic blocks in the path between initial position and the hoisting point
- ▶ limit the number of instructions between the initial position and the beginning of its basic block
- ▶ do not hoist GEPs (except at -Os)
- ▶ limit the number of dependent instructions to be hoisted

Knobs

- ▶ **-enable-gvn-hoist:** enable the GVN-hoist pass (default = on)
- ▶ **-Os, -Oz:** allow GEPs to be hoisted independently of ld/st
- ▶ **-gvn-hoist-max-bbs:** max number of basic blocks on the path between hoisting locations (default = 4, unlimited = -1)
- ▶ **-gvn-hoist-max-depth:** hoist instructions from the beginning of the BB up to the maximum specified depth (default = 100, unlimited = -1)
- ▶ **-gvn-hoist-max-chain-length:** maximum length of dependent chains to hoist (default = 10, unlimited = -1)
- ▶ **-gvn-max-hoisted:** max number of instructions to hoist (default unlimited = -1)

GVN-Hoist: Evaluation

- ▶ $< 1\%$ compile time overhead: 1678 \rightarrow 1692 Bn insns (callgrind compiling the test-suite at -O3 on x86_64-linux)
- ▶ more hoists than CFG-simplify: 15048 \rightarrow 25318 (compiling the test-suite for x86_64 at -O3)

Scalars hoisted	8960
Scalars removed	11940
Loads hoisted	16301
Loads removed	22690
Stores hoisted	50
Stores removed	50
Calls hoisted	7
Calls removed	7
Total Instructions hoisted	25318
Total Instructions removed	34687

Code size reduction

Code-size metric (.text)	Number
Total benchmarks	497
Total gained in size	39
Total decrease in size	58
Median decrease in size	2.9%
Median increase in size	2.4%

- ▶ test-suite compiled at -O3 for x86_64-linux
- ▶ increase in size due to more inlining
- ▶ many effects due to early scheduling of the pass

Discussion

- ▶ schedule GVN-hoist pass several times?
- ▶ remove CFGSimplify's hoisting?
- ▶ hoist + sink interactions (discuss with James Molloy)
- ▶ early scheduling in opt needs tuning with target info?
- ▶ make GVN-hoist more aggressive for -Os and -Oz?
- ▶ need a better GVN implementation?
- ▶ Memory-SSA is easy to use and fast: **so please use it!**
(thanks Danny, Georges, and others)