GVN-Hoist: Hoisting Computations from Branches

Sebastian Pop and Aditya Kumar

SARC: Samsung Austin R&D Center

November 3, 2016
CFG Simplify’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 → 1692 Bn insns (callgrind compiling the test-suite on x86_64-linux)
Original program

\[
i = \frac{1}{d};
\]
\[
\text{if } (i \geq 0) \{ \\
    u = a \times i; \\
    v = b \times i;
\}
\text{else } \{ \\
    u = b \times i; \\
    v = a \times i;
\}
\]
CFGSimplify limits

Original program

```java
i = 1/d;
if (i >= 0) {
    u = a * i;
    v = b * i;
} else {
    u = b * i;
    v = a * i;
}
```

Expressions hoisted

```java
i = 1/d;
x = a * i;
y = b * i;
if (i >= 0) {
    u = x;
    v = y;
} else {
    u = y;
    v = 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

\[ a = x + y \]
\[ b = x + 1 \]
\[ c = y + 1 \]
\[ d = b + c \]
\[ e = a + 2 \]
\[ f = \text{load} \; d \]
\[ g = \text{load} \; e \]
GVN: Value Numbering Example and Limitations

Simple program

\[
\begin{align*}
a &= x + y \\
b &= x + 1 \\
c &= y + 1 \\
d &= b + c \\
e &= a + 2 \\
f &= \text{load } d \\
g &= \text{load } e
\end{align*}
\]

Value Numbering

\[
\begin{align*}
(a, 1) \\
(b, 2) \\
(c, 3) \\
(d, 4) \\
e, 4)
\end{align*}
\]

Limitations to current GVN implementation
GVN: Value Numbering Example and Limitations

Simple program

\[
\begin{align*}
    a &= x + y \\
    b &= x + 1 \\
    c &= y + 1 \\
    d &= b + c \\
    e &= a + 2 \\
    f &= \text{load } d \\
    g &= \text{load } e
\end{align*}
\]

Value Numbering

\[
\begin{align*}
    (a, 1) \\
    (b, 2) \\
    (c, 3) \\
    (d, 4) \\
    (e, 4)
\end{align*}
\]

Limitations to current GVN implementation

\[
\begin{align*}
    (f, 5) \\
    (g, 6) \\
    // \text{ should be } (g, 5)
\end{align*}
\]
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
Cost models

tuned on x86_64 and AArch64 Linux: test-suite, SPEC 2k, 2k6, ...
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 → 1692 Bn insns (callgrind compiling the test-suite at -O3 on x86_64-linux)
- more hoists than CFG-simplify: 15048 → 25318 (compiling the test-suite for x86_64 at -O3)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalars hoisted</td>
<td>8960</td>
</tr>
<tr>
<td>Scalars removed</td>
<td>11940</td>
</tr>
<tr>
<td>Loads hoisted</td>
<td>16301</td>
</tr>
<tr>
<td>Loads removed</td>
<td>22690</td>
</tr>
<tr>
<td>Stores hoisted</td>
<td>50</td>
</tr>
<tr>
<td>Stores removed</td>
<td>50</td>
</tr>
<tr>
<td>Calls hoisted</td>
<td>7</td>
</tr>
<tr>
<td>Calls removed</td>
<td>7</td>
</tr>
<tr>
<td>Total Instructions hoisted</td>
<td>25318</td>
</tr>
<tr>
<td>Total Instructions removed</td>
<td>34687</td>
</tr>
</tbody>
</table>
## Code size reduction

<table>
<thead>
<tr>
<th>Code-size metric (<code>.text</code>)</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total benchmarks</td>
<td>497</td>
</tr>
<tr>
<td>Total gained in size</td>
<td>39</td>
</tr>
<tr>
<td>Total decrease in size</td>
<td>58</td>
</tr>
<tr>
<td>Median decrease in size</td>
<td>2.9%</td>
</tr>
<tr>
<td>Median increase in size</td>
<td>2.4%</td>
</tr>
</tbody>
</table>

- 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)