Short-Circuit Compiler Transformation: Optimizing Conditional Blocks

Mohammad Ali Ghodrat, Tony Givargis, Alex Nicolau

Department of Computer Sciences
Center for Embedded Computing System
University of California, Irvine
Outline

- Introduction and motivational example
- Related work and underlying analysis
- Proposed transformation
- Measuring/Optimizing delay
- Results
- Conclusion
Growing importance of embedded software

Importance of aggressive compiler optimizations for embedded systems

Long time compilation for embedded software is tolerable

Source: The Boston Consulting Group & Thomson Financial Database, 2005
Motivational example

Original

```c
for(x=min; x<max; x++) {
    for(y=min; y<max; y++) {
        if(x*x+y*y == x*x*y )
            color(x, y, BLACK);
    }
}
```

Transformed

```c
// expr is false for y<0
for(x=min; x<max; x++) {
    for(y=min; y<max; y++) {
        if( y < 0 )
            continue;
        else if(x*x+y*y == x*x*y )
            color(x, y, BLACK);
    }
}
```

Bypass evaluation of expressions when possible
Related work

- Lazy evaluation of Boolean expressions
  - Early works
    - H.D. Huskey et. al [1961]
    - B.W. Arden et. al [1962]
  - Ordering Boolean operands
    - M.Z. Hanani [1977]
  - Effect on code size
    - M.H. Clifton [1998]

- Conditional expression evaluation
  - CASES-2005, TVLSI-2006
Underlying Analysis

- Given a conditional expression:
  - E.g., $2x_0 + x_1 + 4 > 0$
- $n$-dimensional space $S$ is a box-shaped region:
  - $[L_0, U_0] \times [L_1, U_1] \times \ldots \times [L_{n-1}, U_{n-1}]$
- Compute, off-line: true, false, & unknown spaces
- Domain space partitioning problem solution is given in figure.
- Membership test: if a point $X(x_0, x_1, \ldots, x_{n-1})$ is in a space $S$:
  - $(L_0 \leq x_0 \leq U_0) \& \& (L_1 \leq x_1 \leq U_1) \& \& \ldots \& \& (L_{n-1} \leq x_{n-1} \leq U_{n-1})$
Transformation overview

Original

```c
if( Cexpr )
    St_{\text{then}}
else
    St_{\text{else}}
```

Transformed

```c
if( x_0 > 0 \&\& x_0 < 5 )
    St_{\text{then}}
else if( ... )
    St_{\text{else}}
...
else {
    if( Cexpr )
        St_{\text{then}}
    else
        St_{\text{else}}
}
```
Transformation template

if (C )
  St
 else
  St

if (X∈S )
  St
 else if (X∈S )
  St
 else if (X∈S )
  St
 else {

}

St = \begin{cases} 
  St ; \text{ BVi = true} \\
  St ; \text{ BVi = false} 
\end{cases}

Original

Transform
Optimization Strategy

- Pick any conditional block
  - Analyze it
  - Estimate cost of bypasses
  - Estimate cost of conditional block
  - Make decision and mark
  - Repeat until all visited

- Use profiling to improve the estimates above
Overall flow

1. **Processor Model**
2. **C/C++ Code**
3. **Identifying a Conditional Block**
   - **Delay Computation (main blocks)**
   - **Domain Space Partitioning & Probability Annotation**
   - **Conditional Block Analysis (Instrumentation/Profiling)**
   - **Bypass-Code Delay Computation**
   - **Benefit Computation & Transformation**

4. **Transformed code**
for (j=0; j<CBANDS; j++) {
    for (i=0; i<CBANDS; i++) {
        t1[i][j] += 0.474;
        t3 = 15.811389 + 7.5*t1[i][j] - 17.5*sqrt((double) (1.0 + t1[i][j]*t1[i][j]));
        if (t3 <= -100) {
            s[i][j] = 0;
        } else {
            t3 = (t2[i][j] + t3)*LN_TO_LOG10;
            s[i][j] = exp(t3);
        }
    }
}

* Output of the code segment is s[CBANDS][CBANDS]*
Example (Conditional Block Analysis)

\[ C : 15.8 + 7.5*(t1[i][j] + 0.474) - 17.5*\sqrt{1 + (t1[i][j] + 0.474)^2} \leq -100 \]

<table>
<thead>
<tr>
<th>Space</th>
<th>Boolean Value (BV)</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-4.7, 11.8]</td>
<td>False</td>
<td>0.475</td>
</tr>
<tr>
<td>[-30, -4.7]</td>
<td>True</td>
<td>0.312</td>
</tr>
<tr>
<td>[11.8, 30]</td>
<td>True</td>
<td>0.160</td>
</tr>
</tbody>
</table>
```
Example (transformation)

<table>
<thead>
<tr>
<th>Interval</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-4.7, 11.862]</td>
<td>False</td>
</tr>
<tr>
<td>[-30, -4.702]</td>
<td>True</td>
</tr>
</tbody>
</table>

else

if ( t1[i][j] <= 11.862 && t1[i][j] >= -4.7 )
{
    t1[i][j] += 0.474;
    t3 = 15.811389+7.5*t1[i][j]-17.5*sqrt((double) (1.0+t1[i][j]*t1[i][j]));
    t3 = (t2[i][j] + t3)*LN_TO_LOG10;
    s[i][j] = exp(t3);
}
else if ( t1[i][j] <= -4.702 && t1[i][j] >= -30 )
{
    s[i][j] = 0;
}
else if ( t1[i][j] >= 11.864 && t1[i][j] <= 30 )
{
    s[i][j] = 0;
}
else
{
    t1[i][j] += 0.474;
    t3 = 15.811389+7.5*t1[i][j]-17.5*sqrt((double) (1.0+t1[i][j]*t1[i][j]));
    if(t3 <= -100)
    {
        s[i][j] = 0;
    }
    else {
        t3 = (t2[i][j] + t3)*LN_TO_LOG10;
        s[i][j] = exp(t3);
    }
}
```

for(j=0; j<CBANDS; j++) {
    for(i=0; i<CBANDS; i++) {
        if (t1[i][j] <= 11.862 && t1[i][j] >= -4.7) {
            t1[i][j] += 0.474;
            t3 = 15.811389 + 7.5*t1[i][j] - 17.5*sqrt((double) (1.0 + t1[i][j]*t1[i][j]));
            t3 = (t2[i][j] + t3)*LN_TO_LOG10;
            s[i][j] = exp(t3);
        } else if (t1[i][j] <= -4.702 && t1[i][j] >= -30) {
            s[i][j] = 0;
        } else if (t1[i][j] >= 11.864 && t1[i][j] <= 30) {
            s[i][j] = 0;
        } else {
            t1[i][j] += 0.474;
            t3 = 15.811389 + 7.5*t1[i][j] - 17.5*sqrt((double) (1.0 + t1[i][j]*t1[i][j]));
            if(t3 <= -100) {
                s[i][j] = 0;
            } else {
                t3 = (t2[i][j] + t3)*LN_TO_LOG10;
                s[i][j] = exp(t3);
            }
        }
    }
}
Benefit of transformation (delay)

\[
T_{\text{new}} < T_{\text{original}}
\]

\[
T_{\text{original}} = C_{\text{expr}} \cdot \text{delay}
\]

\[
+ \ Prob_{\text{then}} \times S_{\text{then}} \cdot \text{delay}
\]

\[
+ \ Prob_{\text{else}} \times S_{\text{else}} \cdot \text{delay}
\]

Original

Transform
Benefit of transformation (delay)

\[
T_{\text{new}} = P_1 \cdot (\text{case}_1\cdot \text{delay} + S_{BV1}\cdot \text{delay}) + P_2 \cdot (\text{case}_1\cdot \text{delay} + \text{case}_2\cdot \text{delay} + S_{BV2}\cdot \text{delay}) + \ldots + P_m \cdot (\text{case}_1\cdot \text{delay} + \ldots + \text{case}_m\cdot \text{delay} + S_{BVm}\cdot \text{delay}) + (1-P_1-\ldots-P_m) \cdot T_{\text{original}}
\]
Optimizing if-else chain

- Any or all of the cases may be left out because of the last else block.
- As the number of cases that are added grows, $T_{\text{new}}$ increases.
- Add the next highest probability case as long as $T_{\text{new}}$ is strictly decreasing.
## Experimental Results

<table>
<thead>
<tr>
<th>Code seg.</th>
<th>Application-Function desc.</th>
<th>Conditional expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MESA-Compute the fogged color indexes</td>
<td>$exp(c^2 * z^2) &gt; 1$</td>
</tr>
<tr>
<td>2</td>
<td>MESA-Compute the fogged color</td>
<td>$0 \leq exp(-c^2 * z^2) \leq 1$</td>
</tr>
<tr>
<td>3</td>
<td>MP3-Layer 3 Psych. Analysis</td>
<td>$15.8 + 7.5 * t - 17.5 * \sqrt{1.0 + t^2} \leq -60$</td>
</tr>
<tr>
<td>4</td>
<td>MP3-Psych. Analysis</td>
<td>$15.8 + 7.5 * t1 - 17.5 * \sqrt{1.0 + t1^2} &lt; -100$</td>
</tr>
<tr>
<td>5</td>
<td>Graphics-Check for collision</td>
<td>$x \times x + y * y - x \times x * y == 0$</td>
</tr>
<tr>
<td>6</td>
<td>MPEGDEC Initialize Decoder</td>
<td>$(i &lt; 0), (i &gt; 255)$</td>
</tr>
<tr>
<td>7</td>
<td>MPEGENC-Ver./Hor. Filter, 2:1 Subsample</td>
<td>$(i &lt; 5), (i &lt; 4), (i &lt; 3), (i &lt; 2), (i &lt; 1)$</td>
</tr>
<tr>
<td>8</td>
<td>MP3-Layer 3 Psych. Analysis</td>
<td>$j &lt; sync_flush, j &lt; BLKSIZE$</td>
</tr>
<tr>
<td>9</td>
<td>MP3-Read and align audio data</td>
<td>$j &lt; 64$</td>
</tr>
<tr>
<td>10</td>
<td>MPEG-IDCT Initialize</td>
<td>$(i &lt; -256), (i &gt; 255)$</td>
</tr>
<tr>
<td>11</td>
<td>MPEGDEC-Ver./Hor. Interpolation Filter</td>
<td>$(i &lt; 2), (i &lt; 1)$</td>
</tr>
</tbody>
</table>
SPARC – Execution time gain

Max: 88.1%
Min: 8.7%
Avg: 35.1%
SPARC – Code size increase

Max: 158%
Min: -48%
Avg: 36 %
ARM - Execution time gain

Max: 93 %
Min: 1 %
Avg: 36 %
ARM – Power reduction

Max: 93 %
Min:  0 %
Avg: 36 %
Conclusion

- Short-circuit code transformation technique
- Conditional expression can be statically analyzed
- Bypass the conditional expression evaluation.
- Profile-based optimizations
- Optimization applies to some regions of the code
- For an embedded software this increase in compile time is justified
THANKS