Merging Similar Control-Flow Regions in LLVM for Performance and Code Size Benefits

Charitha Saumya (Presenter), Kirshanthan Sundararajah, Milind Kulkarni

LLVM Developers’ Meeting 2022
November 7th – 10th 2022
Similar Code inside Conditional Branches is Plentiful

if ( fixed_mult_quo(bx, ay, by) < ax ) {
    left.end.x = px + bx, left.end.y = py + by;
    right.end.x = px + ax, right.end.y = py + ay;
    code = (*fill_trapezoid)(dev, &left, &right, py, ym, false, pdevc, lop);
    right.start = right.end;
    right.end = left.end;
} else {
    left.end.x = px + ax, left.end.y = py + ay;
    right.end.x = px + bx, right.end.y = py + by;
    code = (*fill_trapezoid)(dev, &left, &right, py, ym, false, pdevc, lop);
    left.start = left.end;
    left.end = right.end;
}

Source: mibench/ghostscript benchmark

Identical/similar operation sequences inside if and else sections

if ((tid & k) == 0) {
    if (shared[tid] > shared[ixj])
        swap(shared[tid], shared[ixj]);
} else {
    if (shared[tid] < shared[ixj])
        swap(shared[tid], shared[ixj]);
}

Source: CUDA implementation of bitonic sort

Similar control–flow regions inside if and else sections
Code Size Reduction

\[
\begin{align*}
%A : & \quad %a = \text{add} \ %b, 10 \\
& \quad %c = \text{mul} \ %a, 20 \\
& \quad \ldots
\end{align*}
\]

\[
\begin{align*}
%B : & \quad %p = \text{add} \ %q, 10 \\
& \quad %r = \text{mul} \ %p, 20 \\
& \quad \ldots
\end{align*}
\]

\[
\text{Size reduction} = \text{cost(add)} + \text{cost(mul)} - \text{cost(select)}
\]
Reducing Control-flow Divergence

Contains a Divergent Branch

- Thread group (warp/wavefront)
Reducing Control-flow Divergence

%B:
...
%a = load ..
%a = mul ..
%b = div ..
...

%C:
...
%p = load ..
%q = mul ..
%r = div ..
...

A \rightarrow B \rightarrow C \rightarrow D

A \rightarrow B \rightarrow \rightarrow C \rightarrow D

\text{time}
Reducing Control-flow Divergence

%B:
...
%a = load ...
%a = mul ....
%b = div ....
....

%C:
...
%p = load ..
%q = mul ....
%r = div ....
....

Convergent execution of common instructions
Code Sinking and Code Hoisting

Code Sinking

Identical instruction sequences are moved to common successor
Code Sinking and Code Hoisting

Code Hoisting

Identical instruction sequences are moved to common predecessor
Branch Fusion

These techniques are limited to exploiting similarity at basic block level (e.g., diamond shaped control-flow)

Control-Flow Melding (CFM)

Control-flow Regions with Similar computations

Control–Flow Melding

Melded control-flow
Program Structure Tree

- Represents all Single-Entry Single-Exit (SESE) regions in a CFG
- Can be obtained using the RegionInfo interface in LLVM
Meldable Regions

- Two SESE regions can be *melded* if,
  - Dominated by a conditional branch
  - No path exists that goes through both the SESE regions
  - Entry blocks of the regions must post-dominate either the left or right successor of the conditional branch
  - They are *isomorphic* (have same control-flow signature)
Region Alignment

- Multiple isomorphic regions in if and else paths?
- Regions are aligned based on Melding Profitability
- Melding Profitability: metric that measures the similarity of two regions based on instruction frequencies

Left regions: A-B, B-C
Right Regions: D-E, E-F, F-G

Region Alignment: A-B with E-F
              B-C with F-G
Instruction Alignment

Aligned region pair

Instruction alignment for A and B computed using instruction compatibility and cost

Aligned region pair

Aligned

Unaligned
Code Generation

Aligned instruction pair

Generated melded instructions

Generated code

Unaligned instructions are executed conditionally
Ensuring Correctness

- Melding can break the def-use chains outside the melded regions

\[
\text{%B:} \quad \ldots \quad \text{%x} = \ldots \\
\text{...} \\
\text{%D:} \\
\text{...} \\
\text{\ldots = mul \text{%x} \ldots} \\
\text{...} \\
\text{%C:} \\
\text{...} \\
\text{%A:} \\
\text{...} \\
\text{%B:} \\
\text{...} \\
\text{\ldots = mul \text{%x} \ldots} \\
\text{...} \\
\text{%C:} \\
\text{...} \\
\text{%A:} \\
\text{...} \\
\text{%B:} \\
\text{...} \\
\text{\ldots = mul \text{%x} \ldots} \\
\text{...} \\
\text{%C:} \\
\text{...} \\
\text{%A:} \\
\text{...}
\]

%\text{x no longer dominates %y !!}
Ensuring Correctness

- Melding can break the def-use chains outside the melded regions
Ensuring Correctness

- Handling PHI instructions
Implementation (CPU Code Size)

- LLVM-IR transformation pass
- Process all if-then-else branches
- Uses \textit{TargetTransformInfo} interface
  - For instruction alignment
  - For evaluating the code size before and after

```
Collect conditional branches

Branches left to process?

Yes

\textit{apply the transformation}

Code size reduced?

Yes

No

Discard the changes

No

Yes

Finish
```
Implementation (GPU Divergence Reduction)

- Process only divergent if-the-else branches
  - Uses LLVM *DivergenceAnalysis*

- Uses a custom instruction latency cost model to evaluate the profitability
Evaluation (GPU Divergence Reduction)

- Speedups on AMD Radeon Pro Vega 20 GPU + AMD Ryzen CPU
- Comparison against both \textit{-O3 (full opts)} and \textit{Branch Fusion with full opts}
- Benchmarks
  1. Bitonic Sort (BIT)
  2. Parallel and Concurrent Merge (PCM)
  3. Mergesort (MS)
  4. LU-decomposition (LUD)
  5. N-Queens (NQU)
  6. Speckle Reducing Anisotropic Diffusion (SRAD)
  7. DCT Quantization (DCT)
Control-flow Melding (CFM)  Branch Fusion

BIT, PCM, NQU : Performance improvement over branch fusion comes from CFM’s ability to meld at region level

+ - Best block size
Evaluation (Code Size Reduction)

- Reduction in the text section of the final binary in x86
- Comparison against \textit{Oz}

- Benchmarks Suites
  - MiBench
  - SPEC 2006
  - SPEC 2017
Code Size Reductions – MiBench

![Bar chart showing code size reductions for various benchmarks. The chart includes benchmarks like blowfish, ghostscript, ispell, jpeg_c, jpeg_d, patricia, ppp, r symlink, tiff2bw, tiff2dither, tiff2median, tiff2rgba, typeset, and mean. The reductions range from 0 to 2000 bytes, with a mean of 34.6 bytes.](image)
Code Size Reductions – SPEC

SPEC06

SPEC17
Summary

● Similar code inside conditional branches is quite common

● Traditional techniques like code hoisting/sinking are not sufficient to fully exploit the code similarity

● We propose Control-Flow Melding (CFM) that merge similar code at region level

● Our LLVM implementation of CFM shows its utility as a general compiler transformation

Code:

https://github.com/charitha22/llvm-project-codesize/tree/cfm-dev

Contact Us:

Charitha Saumya (cgusthin@purdue.edu)

https://charitha22.github.io/
Merging Similar Control-Flow Regions in LLVM for Performance and Code Size Benefits

Charitha Saumya (Presenter), Kirshanthan Sundararajah, Milind Kulkarni

LLVM Developers’ Meeting 2022

November 7th – 10th 2022