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

Charitha Saumya (Presenter), Kirshanthan Sundararajah, Milind Kulkarni

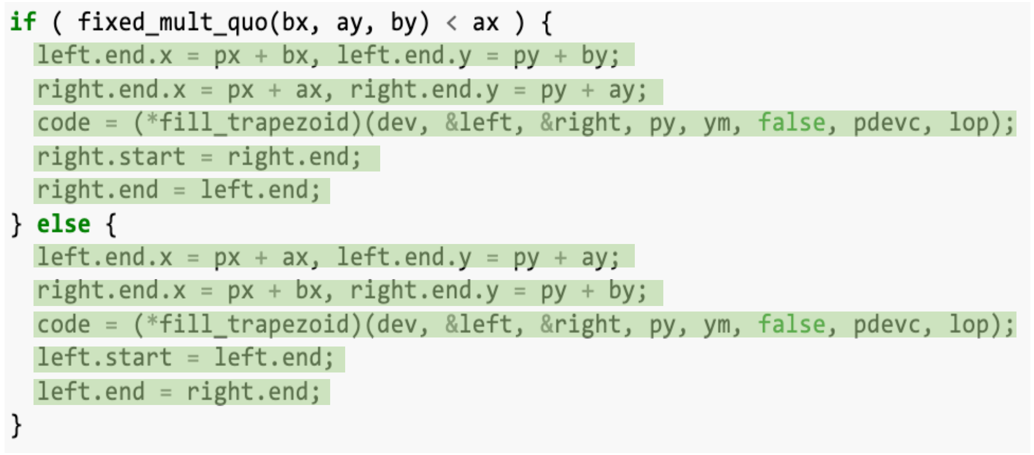
PUR PL

PURDUE UNIVERSITY®

LLVM Developers' Meeting 2022

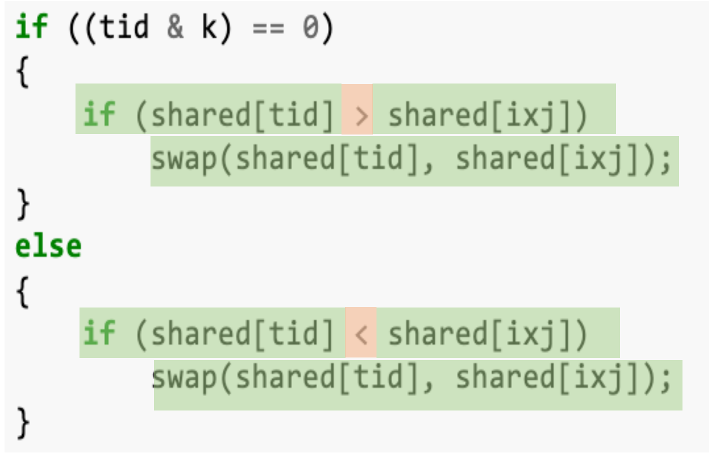
November 7th – 10th 2022

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



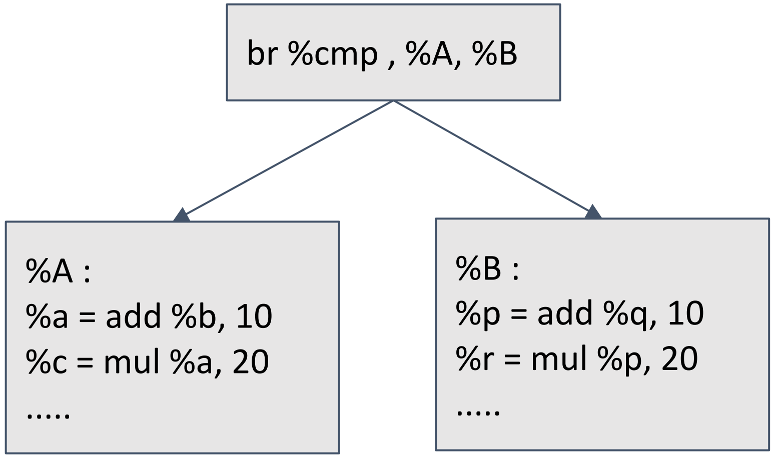
```
if ((tid & k) == 0)
{
    if (shared[tid] > shared[ixj])
        swap(shared[tid], shared[ixj]);
}
else
{
    if (shared[tid] < shared[ixj])
        swap(shared[tid], shared[ixj]);
}
```

Source : *CUDA* implementation of bitonic sort

Identical/similar operation sequences inside if and else sections

Similar control–flow regions inside if and else sections

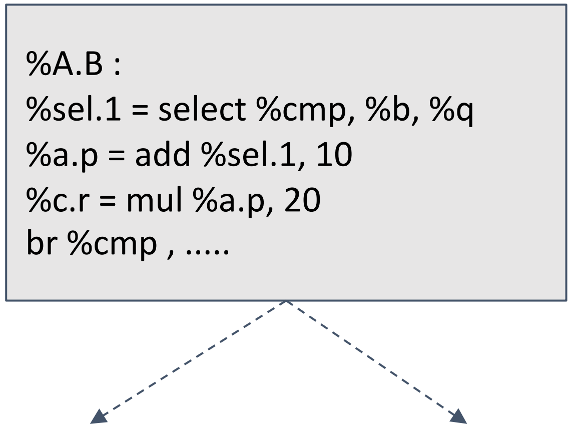
Code Size Reduction



```
br %cmp , %A, %B
```

```
%A :
%a = add %b, 10
%c = mul %a, 20
.....
```

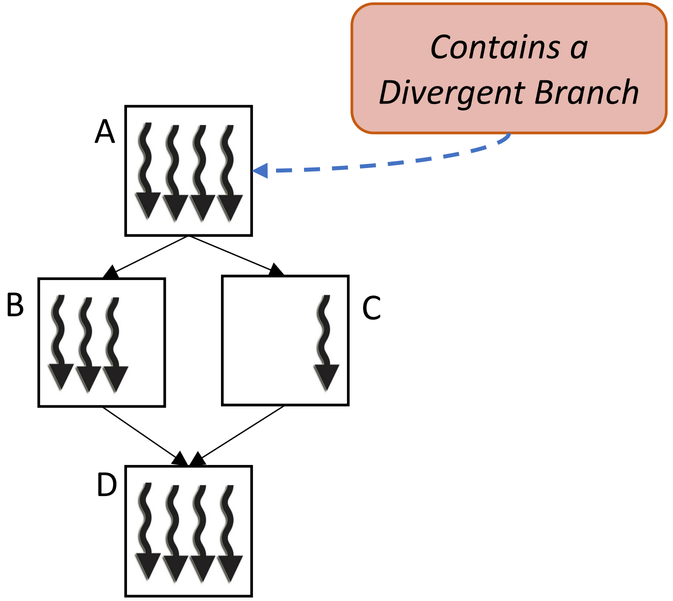
```
%B :
%p = add %q, 10
%r = mul %p, 20
.....
```

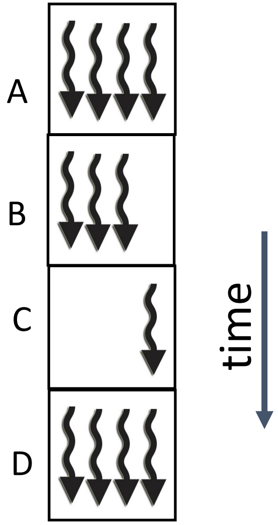


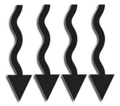
```
%A.B :
%sel.1 = select %cmp, %b, %q
%a.p = add %sel.1, 10
%c.r = mul %a.p, 20
br %cmp , .....
```

Size reduction = cost(add) + cost(mul) - cost(select)

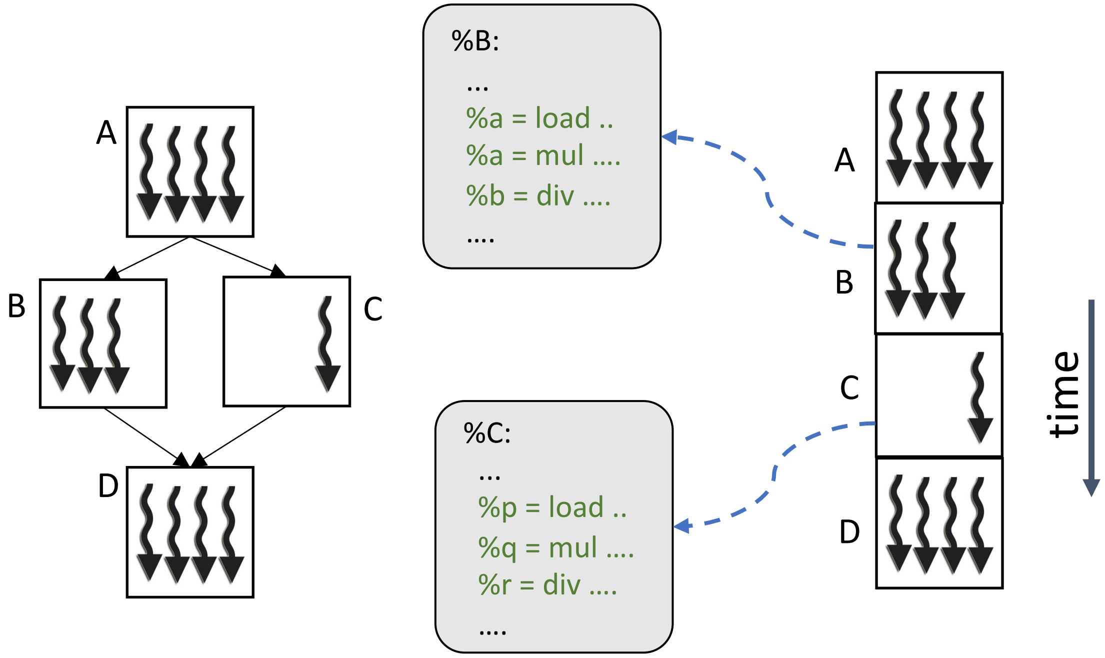
Reducing Control-flow Divergence





 - Thread group (warp/wavefront)

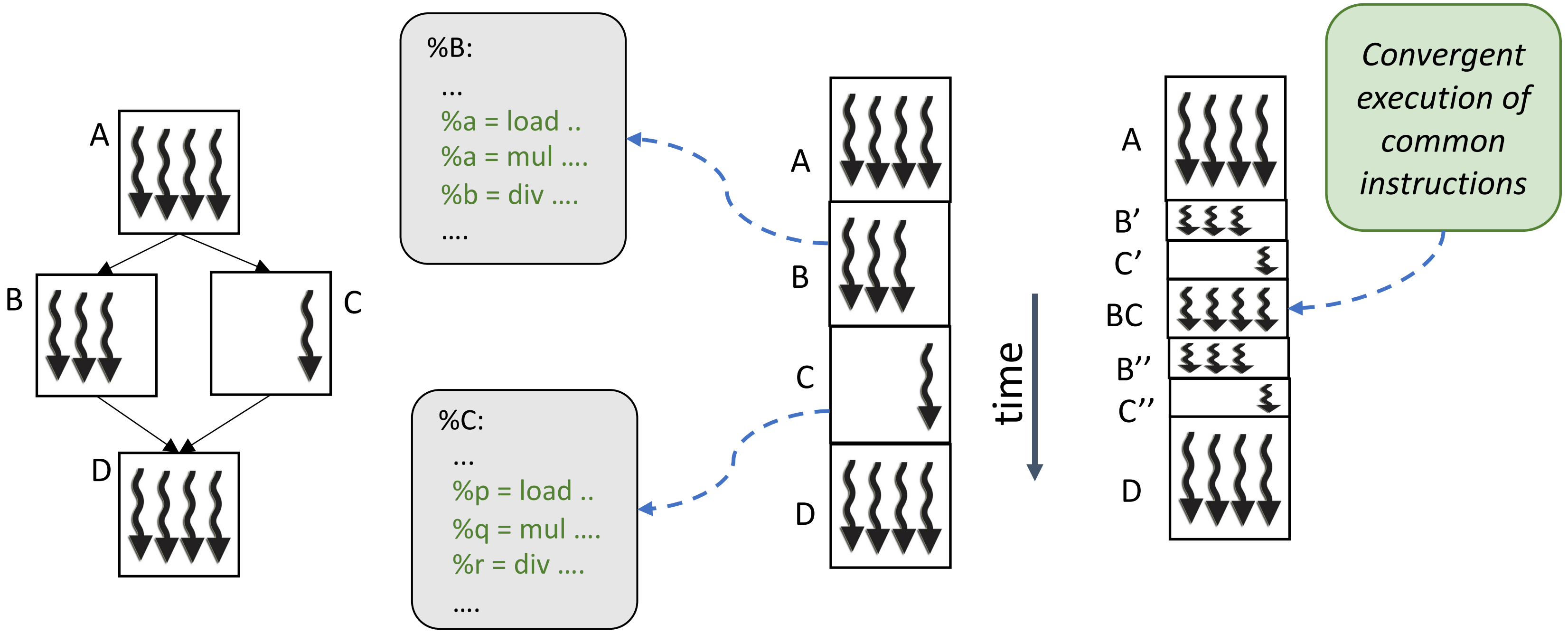
Reducing Control-flow Divergence



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

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

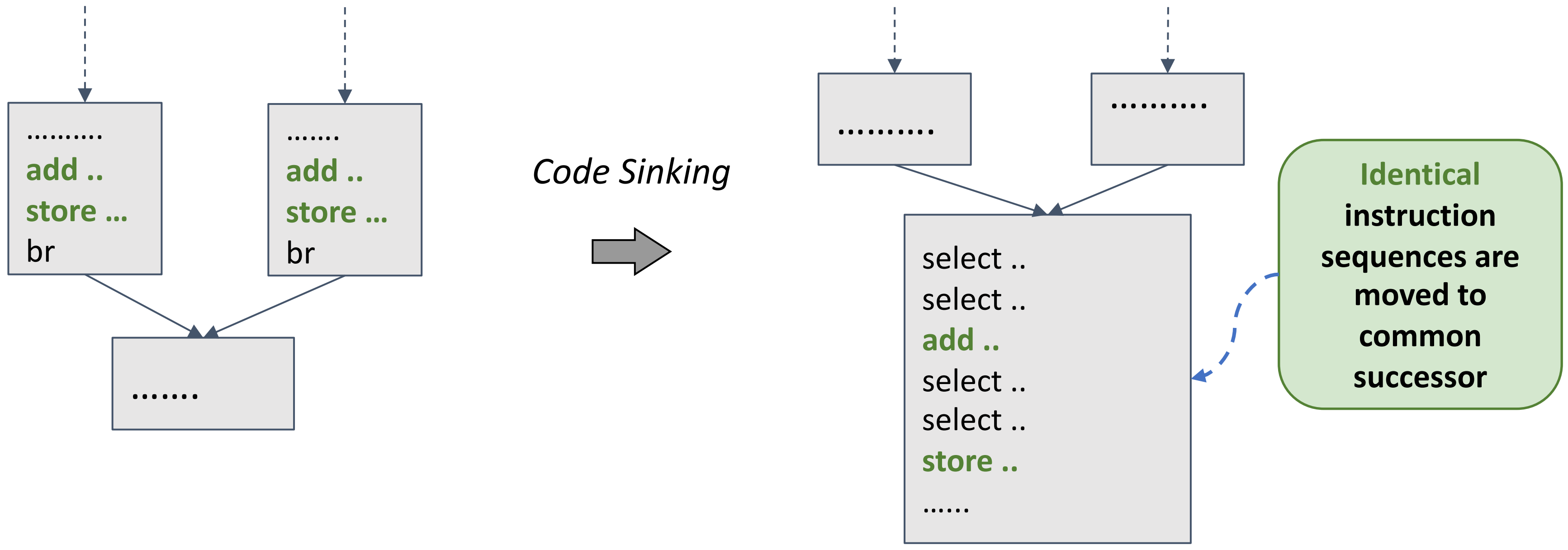
Reducing Control-flow Divergence



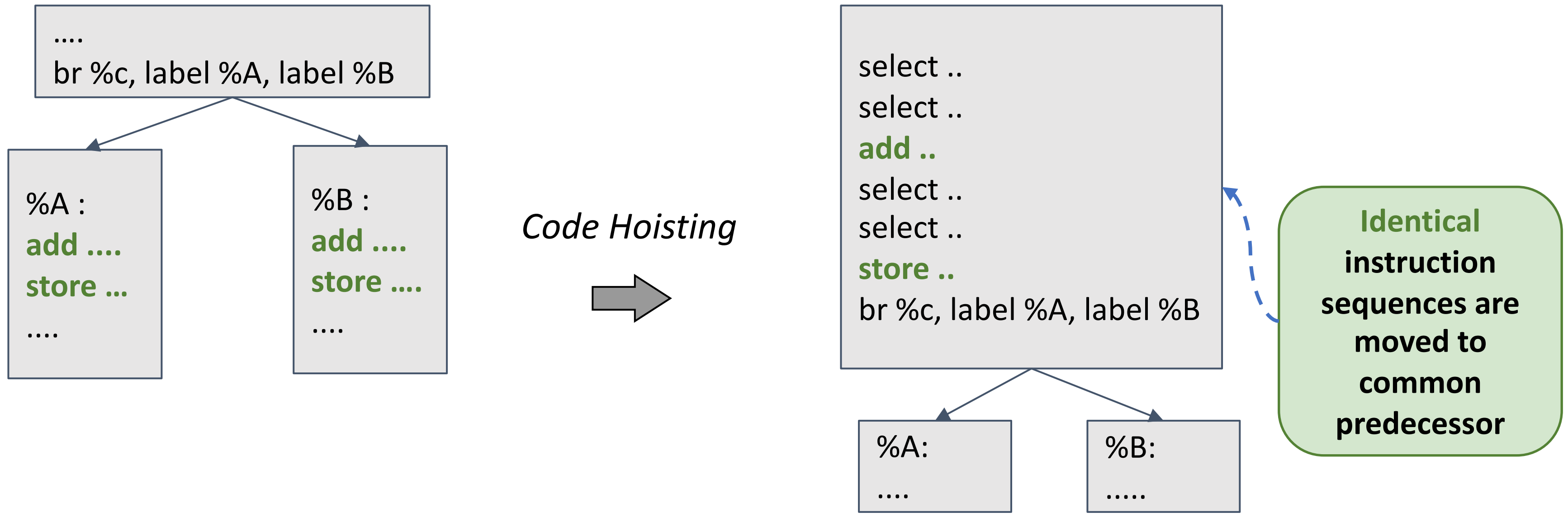
```
%B:
  ...
  %a = load ..
  %a = mul ....
  %b = div ....
  ....
```

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

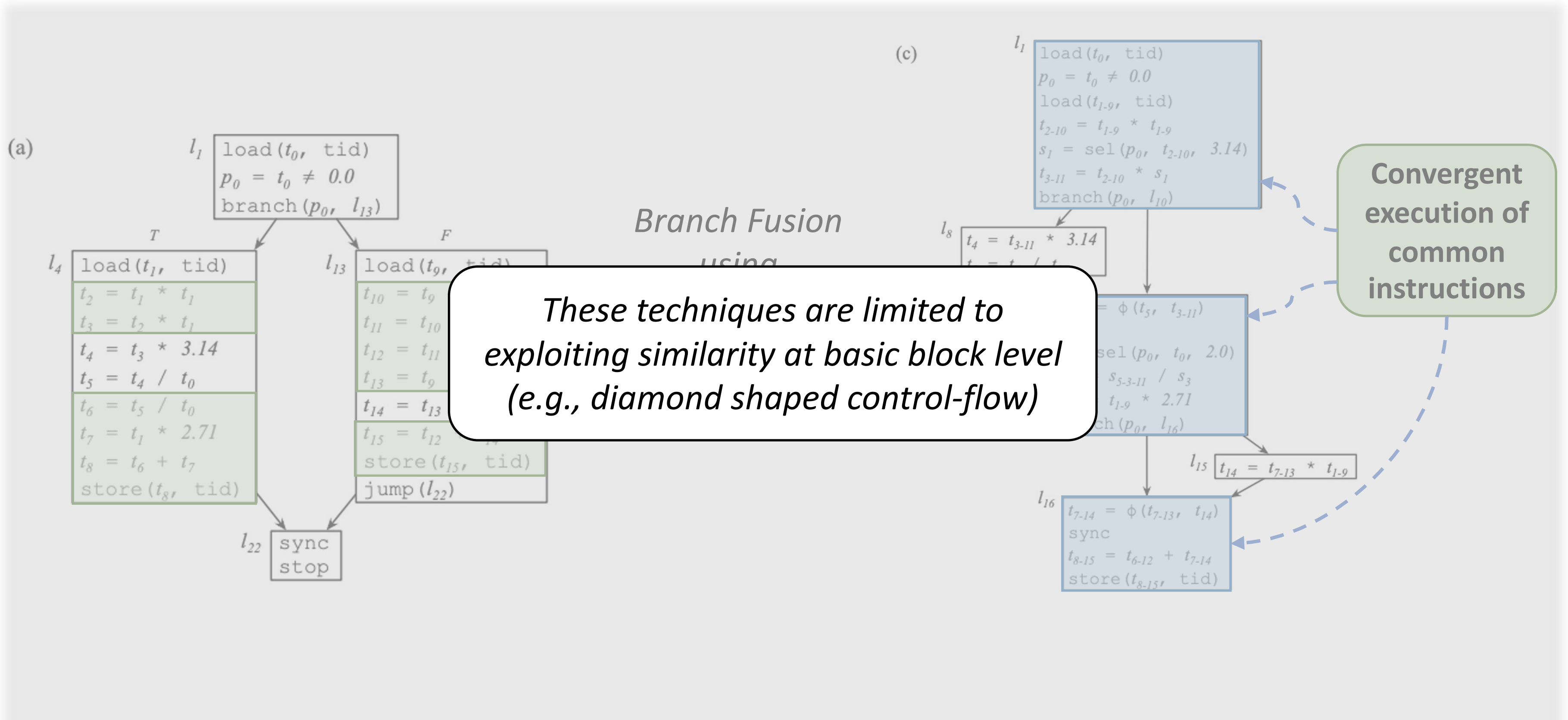
Code Sinking and Code Hoisting



Code Sinking and Code Hoisting



Branch Fusion

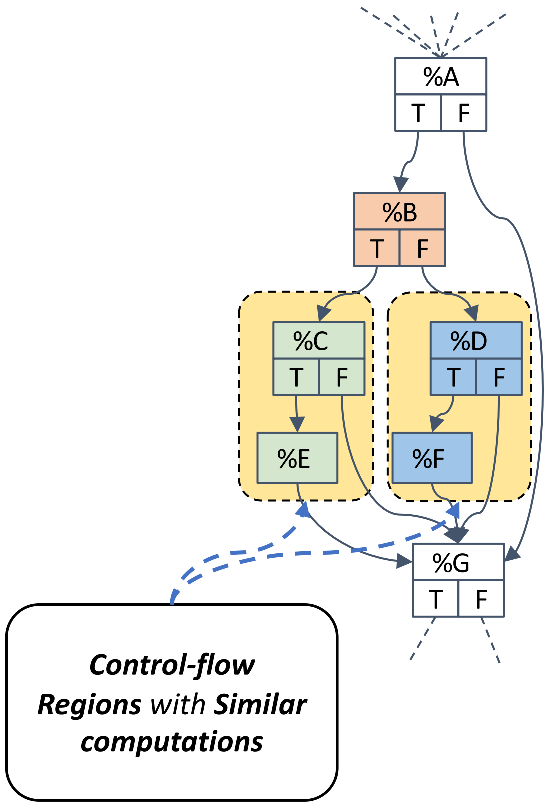


Convergent execution of common instructions

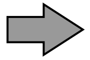
These techniques are limited to exploiting similarity at basic block level (e.g., diamond shaped control-flow)

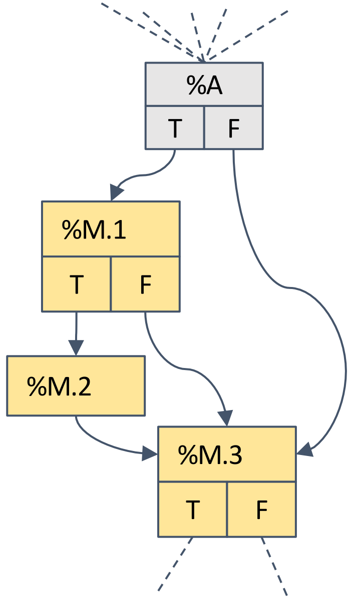
Image Source : B. Coutinho, D. Sampaio, F. M. Q. Pereira and W. Meira Jr., "Divergence Analysis and Optimizations," , PACT'11

Control-Flow Melding (CFM)


%A
T
F
%B
T
F
%C
T
F
%D
T
F
%E
%F
%G
T
F
Control-flow ***Regions*** with ***Similar*** *computations*

Control–Flow Melding

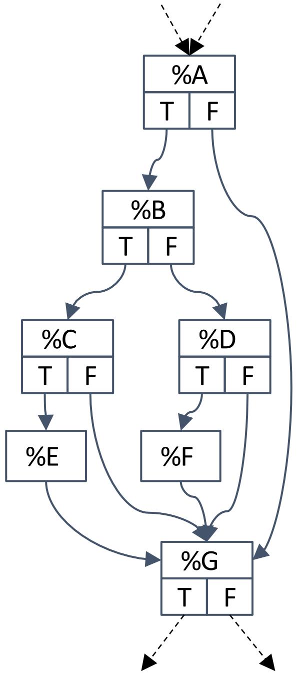


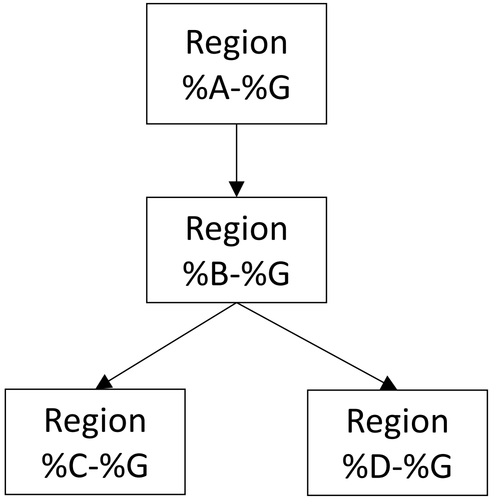


Melded control-flow

Program Structure Tree

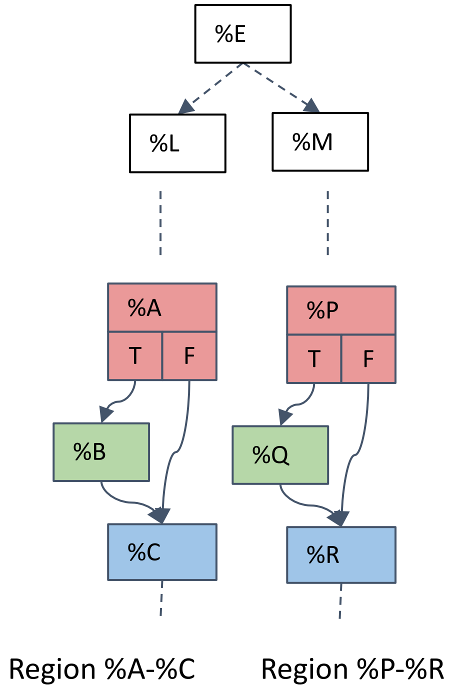
- Represents all Single-Entry Single-Exit (SESE) regions in a CFG
- Can be obtained using the ***RegionInfo*** interface in LLVM





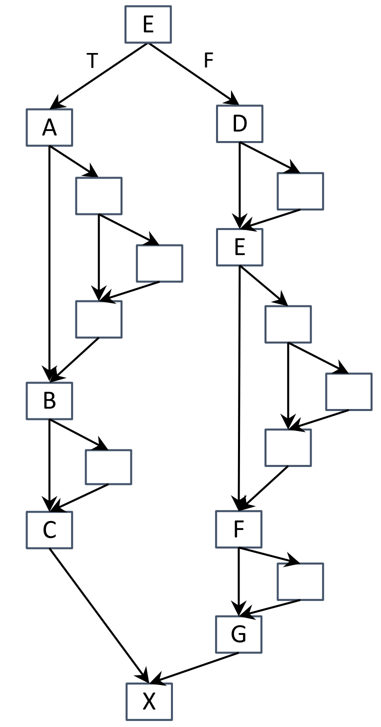
Meldable Regions

- 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)

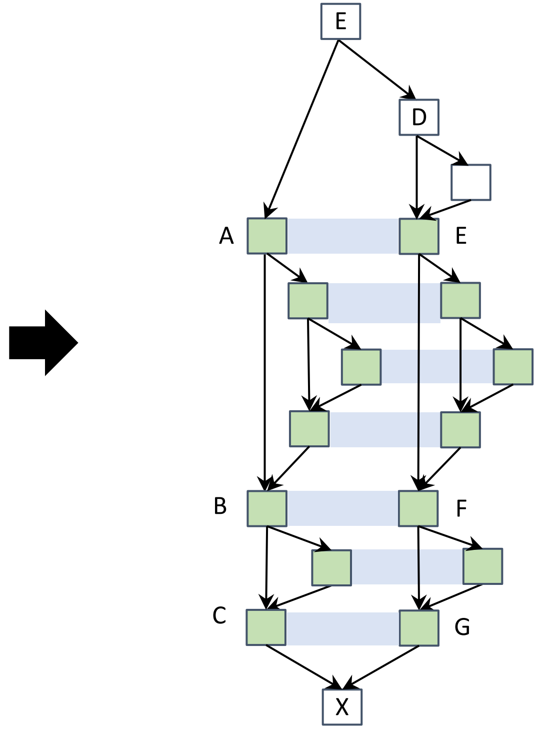


Region Alignment

- 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 base on instruction frequencies

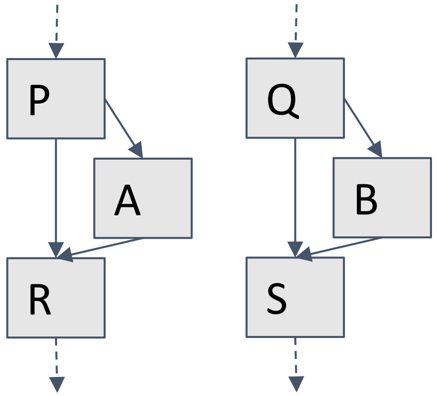


Left regions : *A-B, B-C*
Right Regions : *D-E, E-F, F-G*

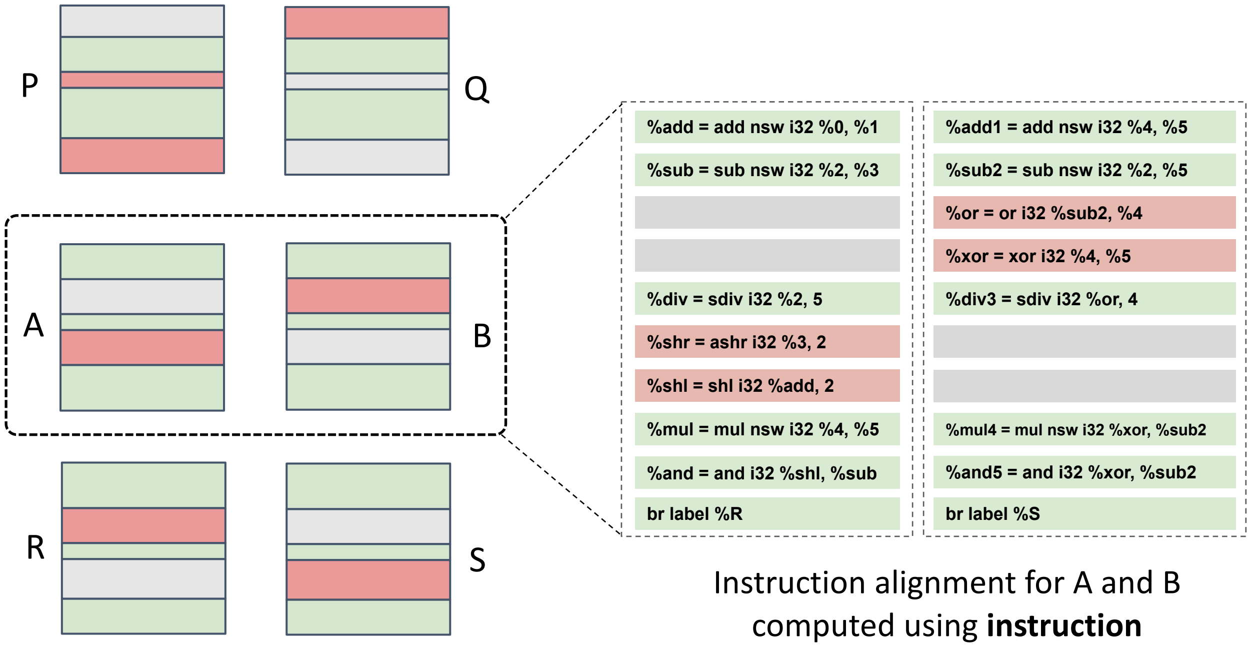


Region Alignment :
A-B with E-F
B-C with F-G

Instruction Alignment



Aligned region pair

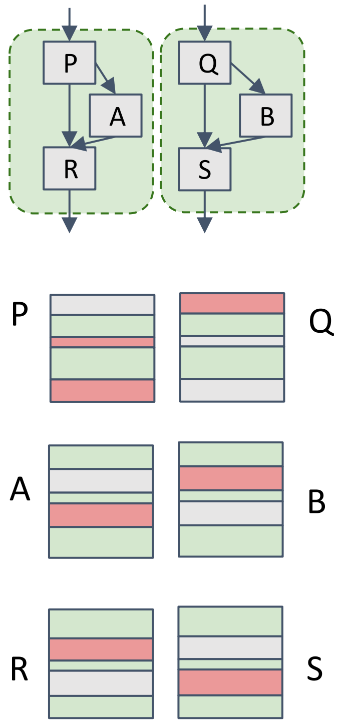


%add = add nsw i32 %0, %1	%add1 = add nsw i32 %4, %5
%sub = sub nsw i32 %2, %3	%sub2 = sub nsw i32 %2, %5
	%or = or i32 %sub2, %4
	%xor = xor i32 %4, %5
%div = sdiv i32 %2, 5	%div3 = sdiv i32 %or, 4
%shr = ashr i32 %3, 2	
%shl = shl i32 %add, 2	
%mul = mul nsw i32 %4, %5	%mul4 = mul nsw i32 %xor, %sub2
%and = and i32 %shl, %sub	%and5 = and i32 %xor, %sub2
br label %R	br label %S

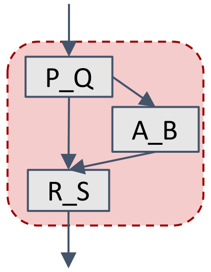
🟩 Aligned 🟥 Unaligned

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

Code Generation



Generated melded control-flow



Generated melded instructions

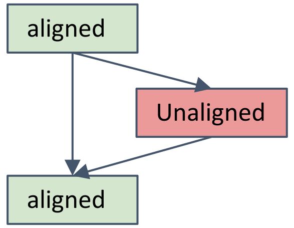
Aligned instruction pair

```
%add = add nsw i32 %0, %1
```

```
%add1 = add nsw i32 %4, %5
```

Generated code

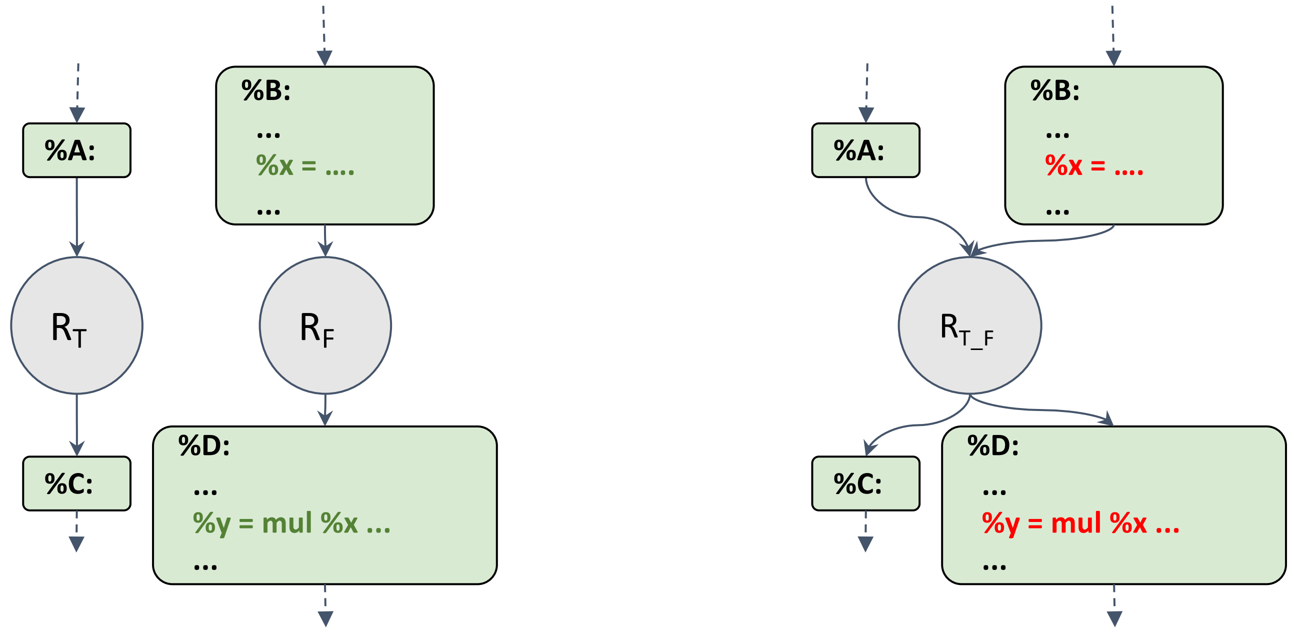
```
%sel1 = select i1 %cmp, i32 %0, i32 %4
%sel2 = select i1 %cmp, i32 %1, i32 %5
 %6 = add nsw i32 %sel1, %sel2
```



Unaligned instructions are executed conditionally

Ensuring Correctness

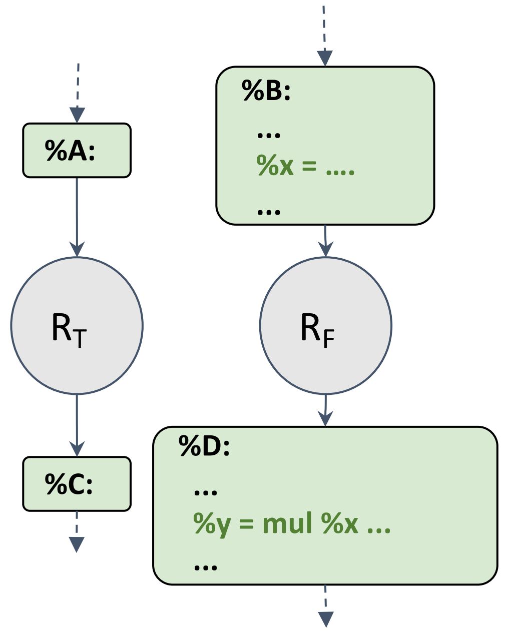
- Melding can break the def-use chains outside the melded regions

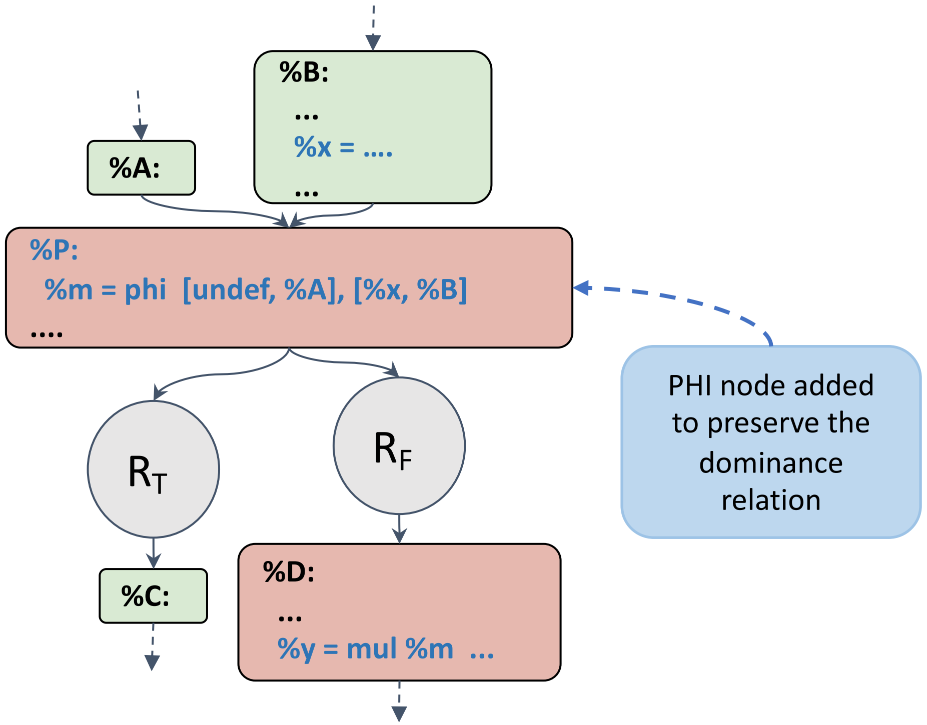


%x no longer dominates %y !!

Ensuring Correctness

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

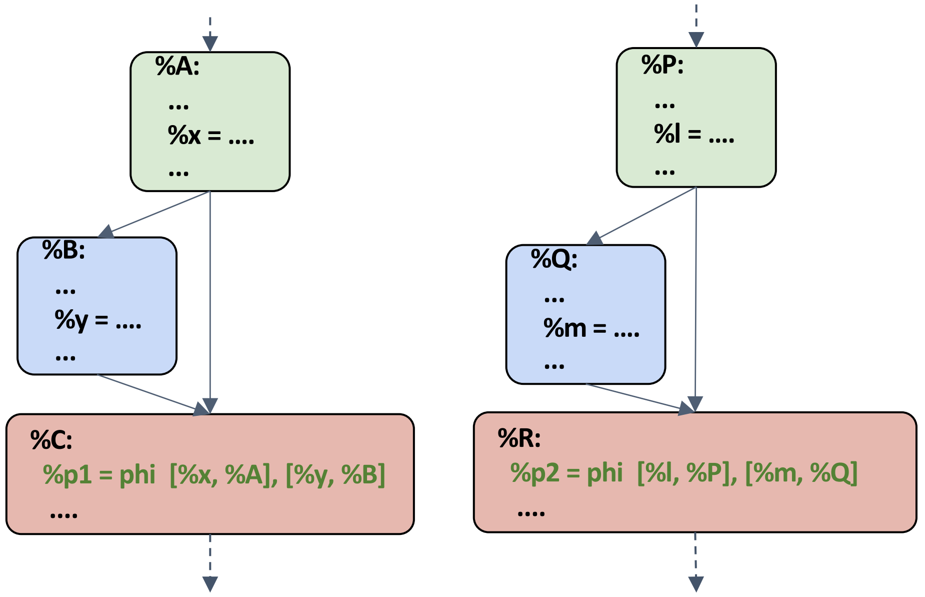


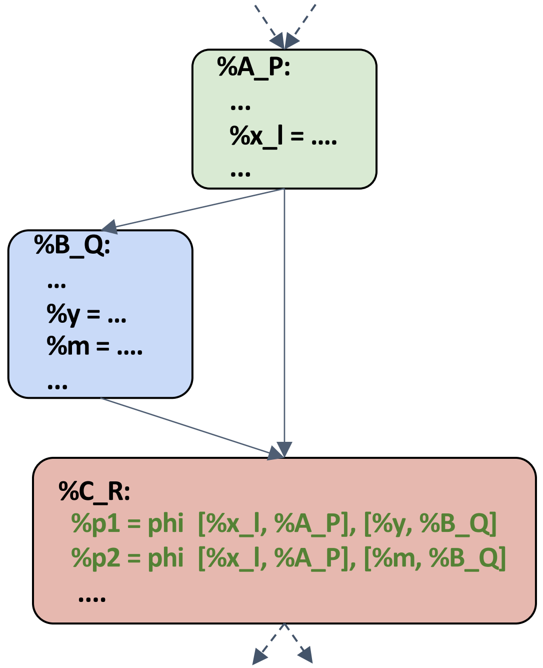


%A:
%B:
...
%x = ...
...
%P:
%m = phi [undef, %A], [%x, %B]
....
R_T
R_F
%C:
%D:
...
%y = mul %m ...
PHI node added to preserve the dominance relation

Ensuring Correctness

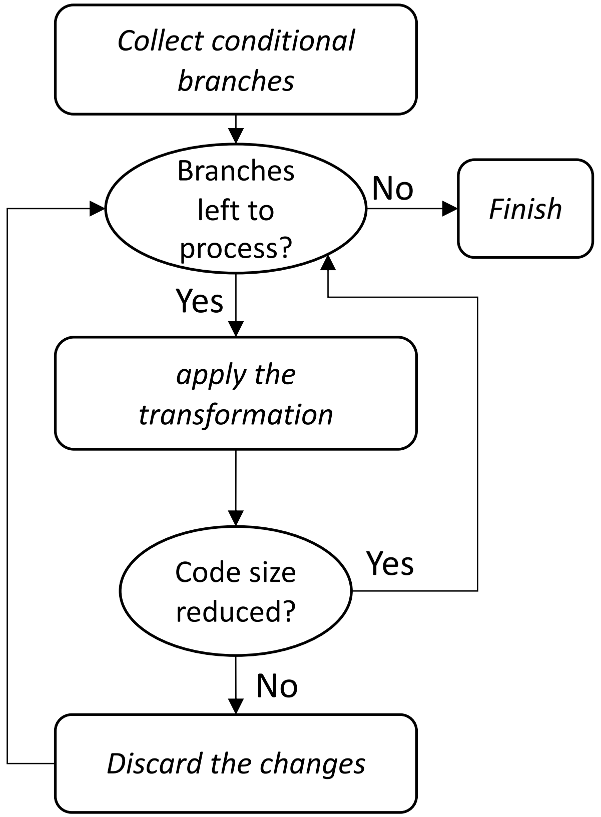
- Handling PHI instructions




%A_P:
...
%x_l = ...
...
%B_Q:
...
%y = ...
%m = ...
...
%C_R:
%p1 = phi [%x_l, %A_P], [%y, %B_Q]
%p2 = phi [%x_l, %A_P], [%m, %B_Q]
....

Implementation (CPU Code Size)

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



Collect conditional branches
Branches left to process?
No
Finish
Yes
apply the transformation
Code size reduced?
Yes
No
Discard the changes

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

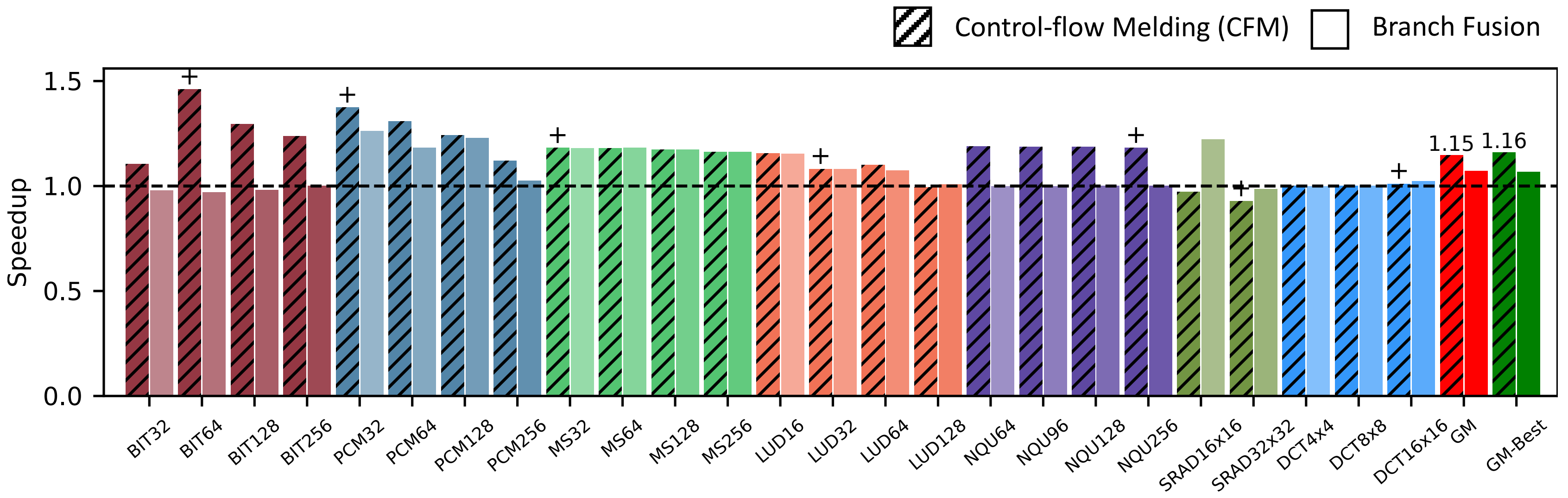
Evaluation (GPU Divergence Reduction)

- Speedups on AMD Radeon Pro Vega 20 GPU + AMD Ryzen CPU

- Comparison against both ***-O3 (full opts)*** and ***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)

Performance



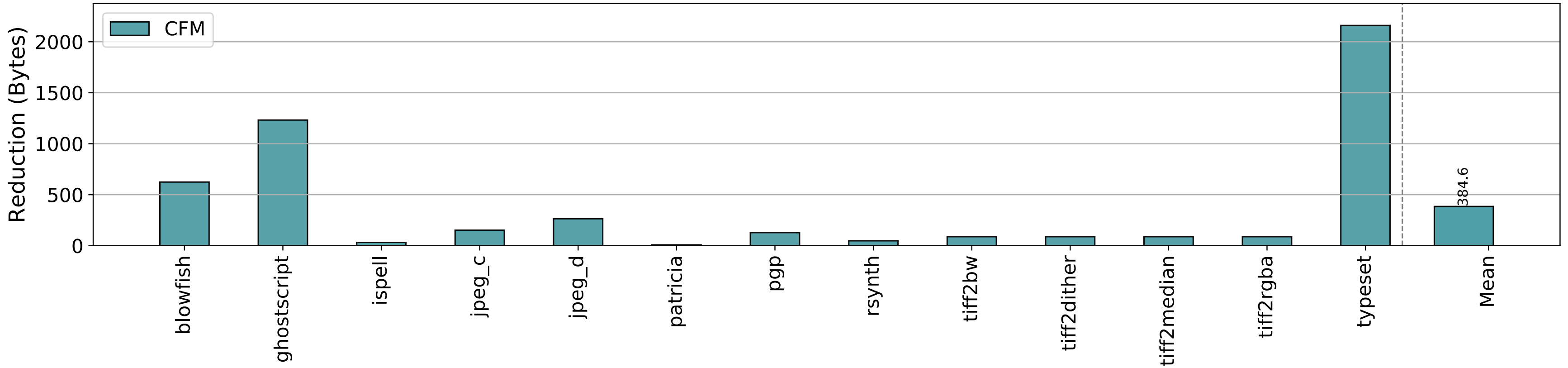
+ - Best block size

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

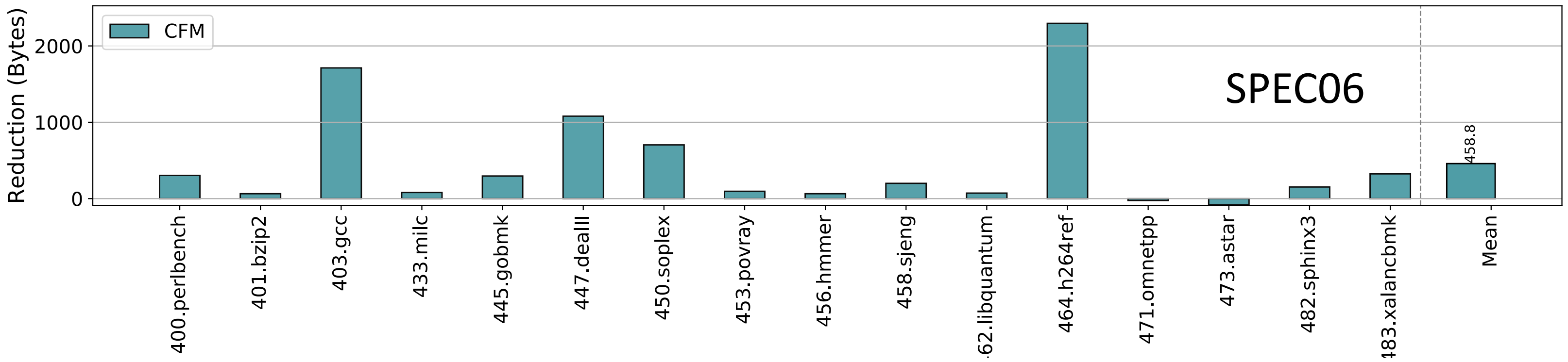
Evaluation (Code Size Reduction)

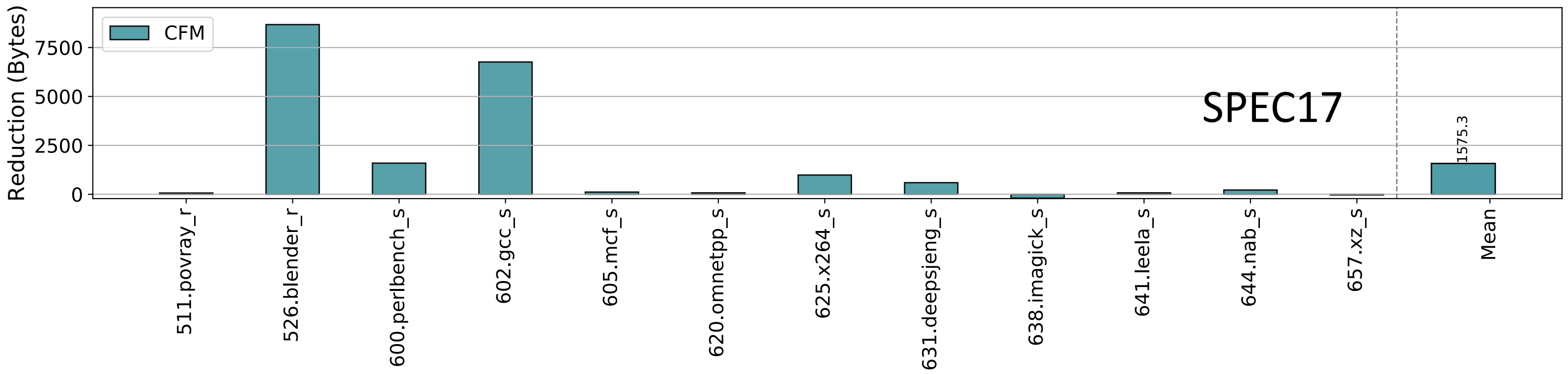
- Reduction in the text section of the final binary in x86
- Comparison against ***–Oz***
- Benchmarks Suites
  - MiBench
  - SPEC 2006
  - SPEC 2017

Code Size Reductions – MiBench



Code Size Reductions – SPEC





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