

DARM: Control-Flow Melding for SIMT Thread Divergence Reduction

Charitha Saumya, Kirshanthan Sundararajah, Milind Kulkarni



CGO '22

April 2nd – 6th 2022

General Purpose Computing on Graphics Processing Units (GPGPU)

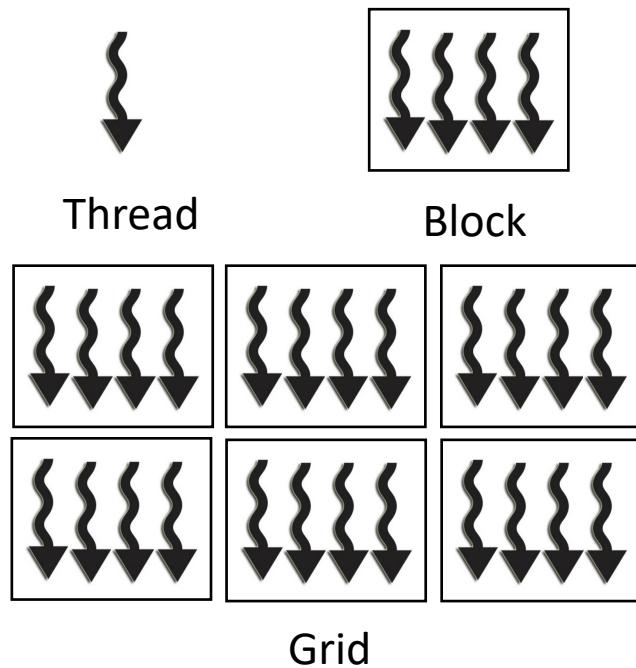
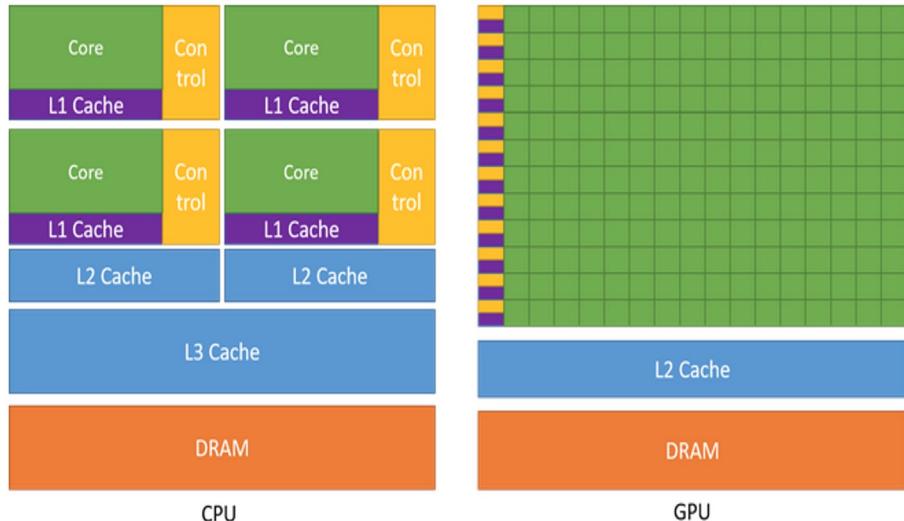


Image Source : <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>

General Purpose Computing on Graphics Processing Units (GPGPU)

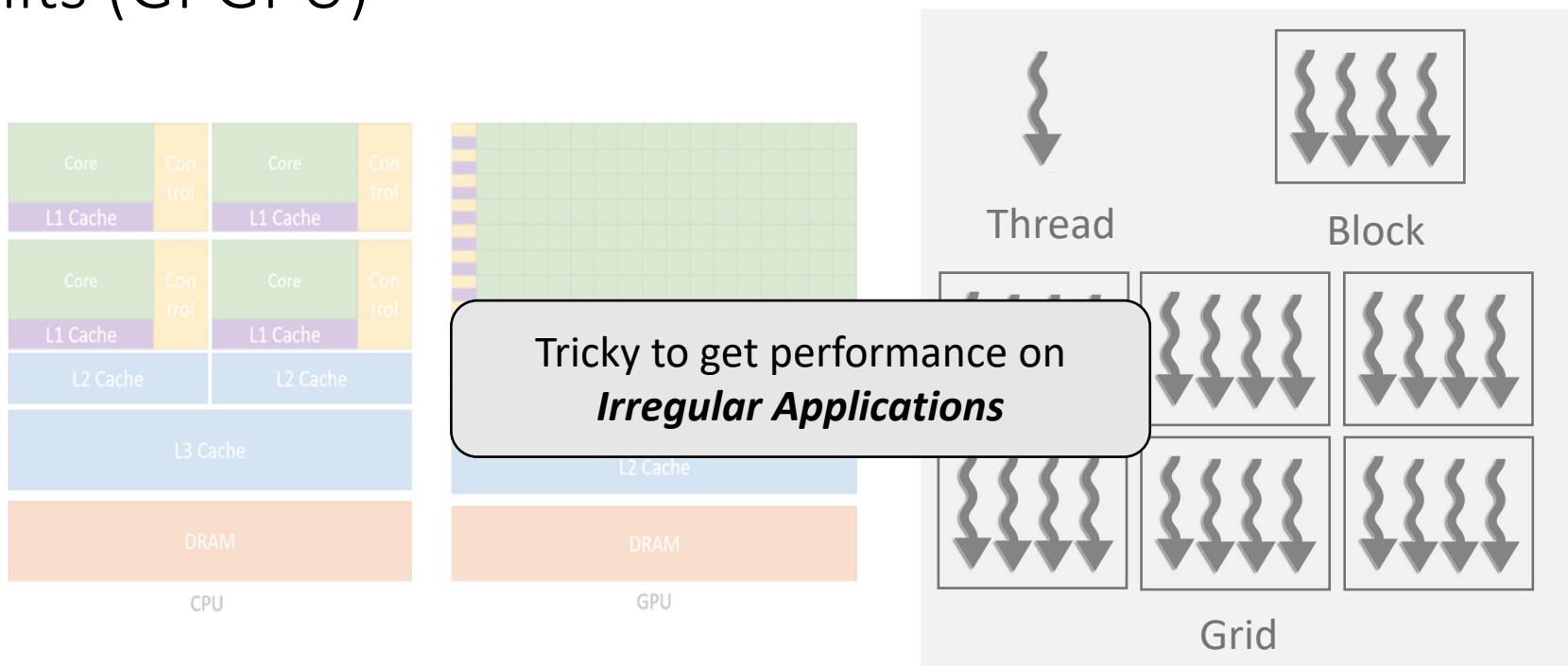
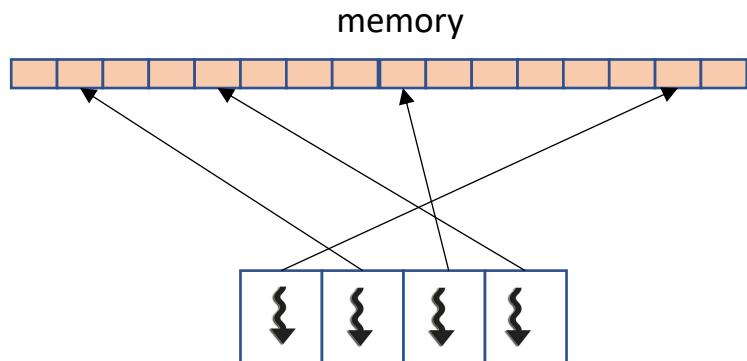


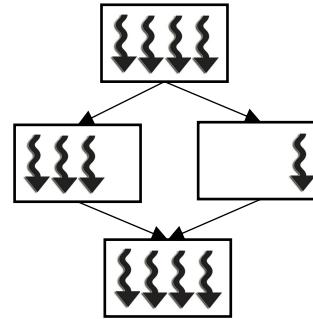
Image Source : <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>

Irregularity in GPU Applications

Memory Divergence

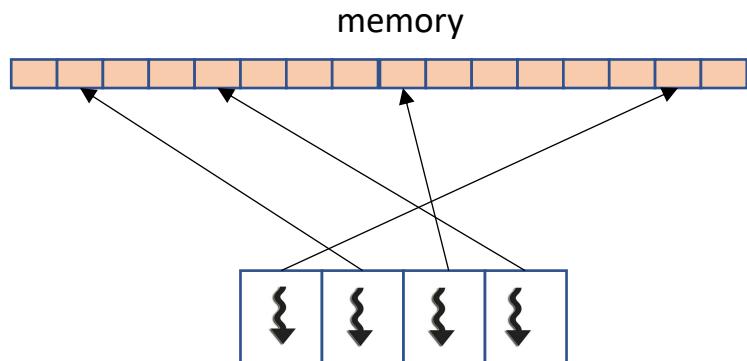


Control-Flow Divergence

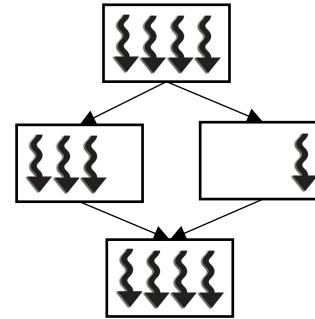


Irregularity in GPU Applications

Memory Divergence

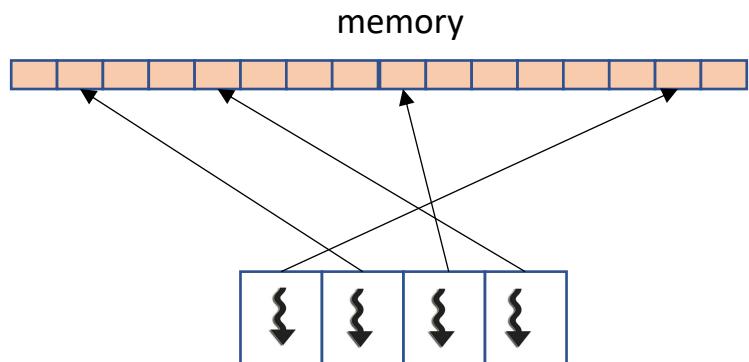


Control-Flow Divergence

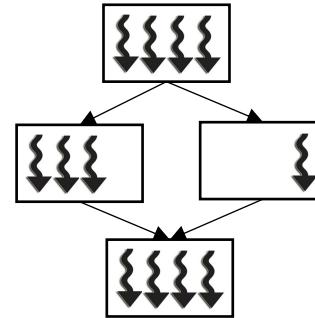


Irregularity in GPU Applications

Memory Divergence



Control-Flow Divergence



Why irregular control-flow is bad for performance?

Single-Instruction-Multiple-Threads (SIMT) Execution Model

- Threads are arranged in groups (warp/wavefront)
- Lockstep execution among threads in a group

Massive Data Parallelism

+

Relatively Energy Efficient

+

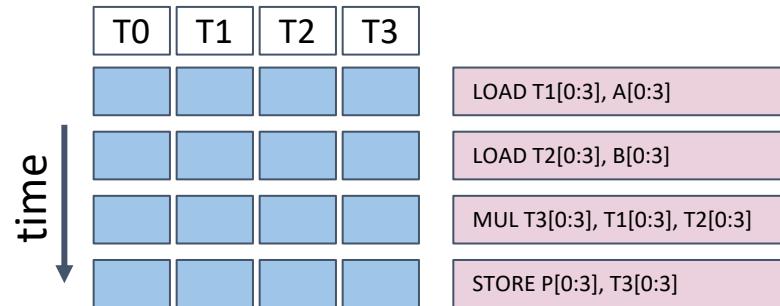
SPMD-style Programming

Single-Instruction-Multiple-Threads (SIMT) Execution Model

$$P[tid] = A[tid] * B[tid]$$

- Threads are arranged in groups (warp/wavefront)
- Lockstep execution among threads in a group

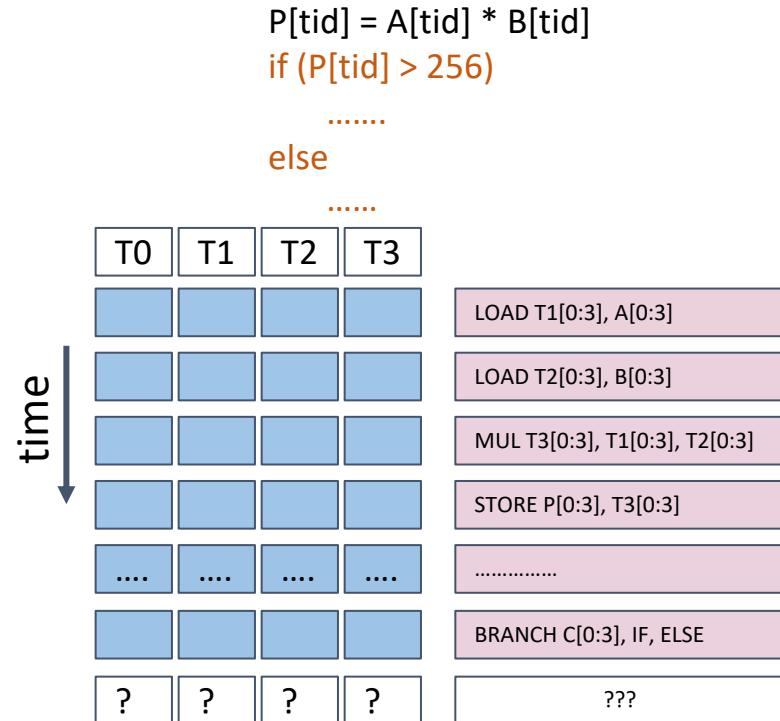
Massive Data Parallelism
+
Relatively Energy Efficient
+
SPMD-style Programming



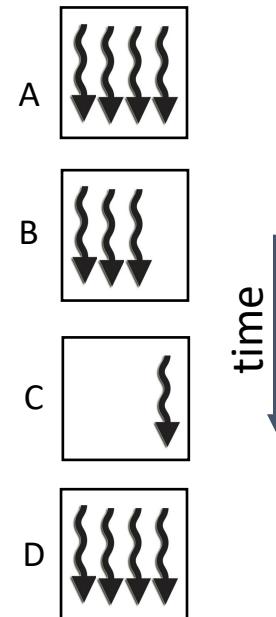
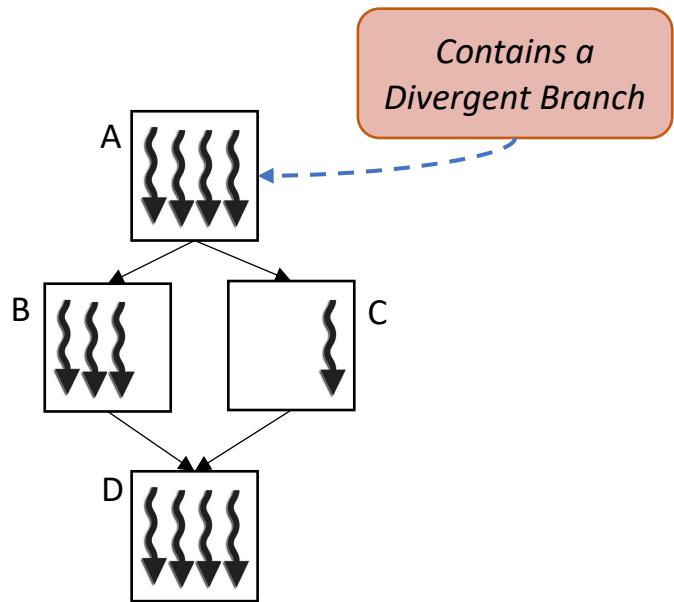
Control-Flow Divergence

- Threads are arranged in groups (warp/wavefront)
- Lockstep execution among threads in a group

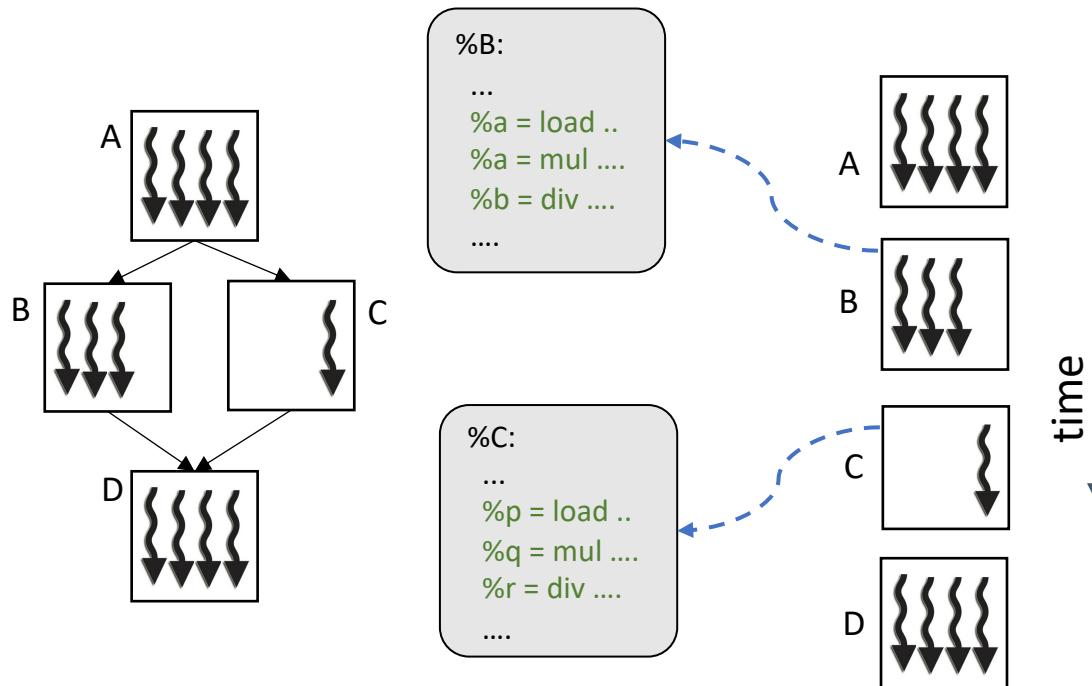
Threads can diverge at branches



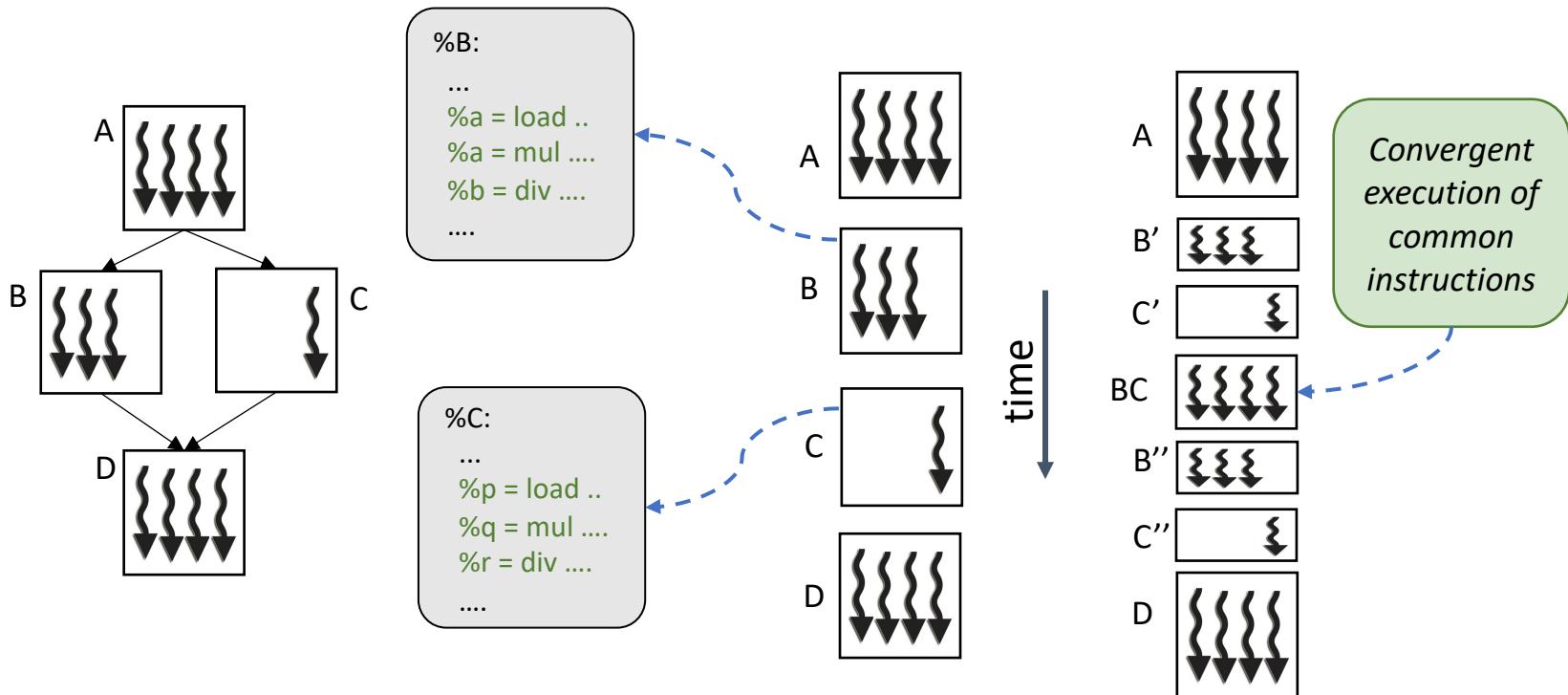
Control-Flow Divergence



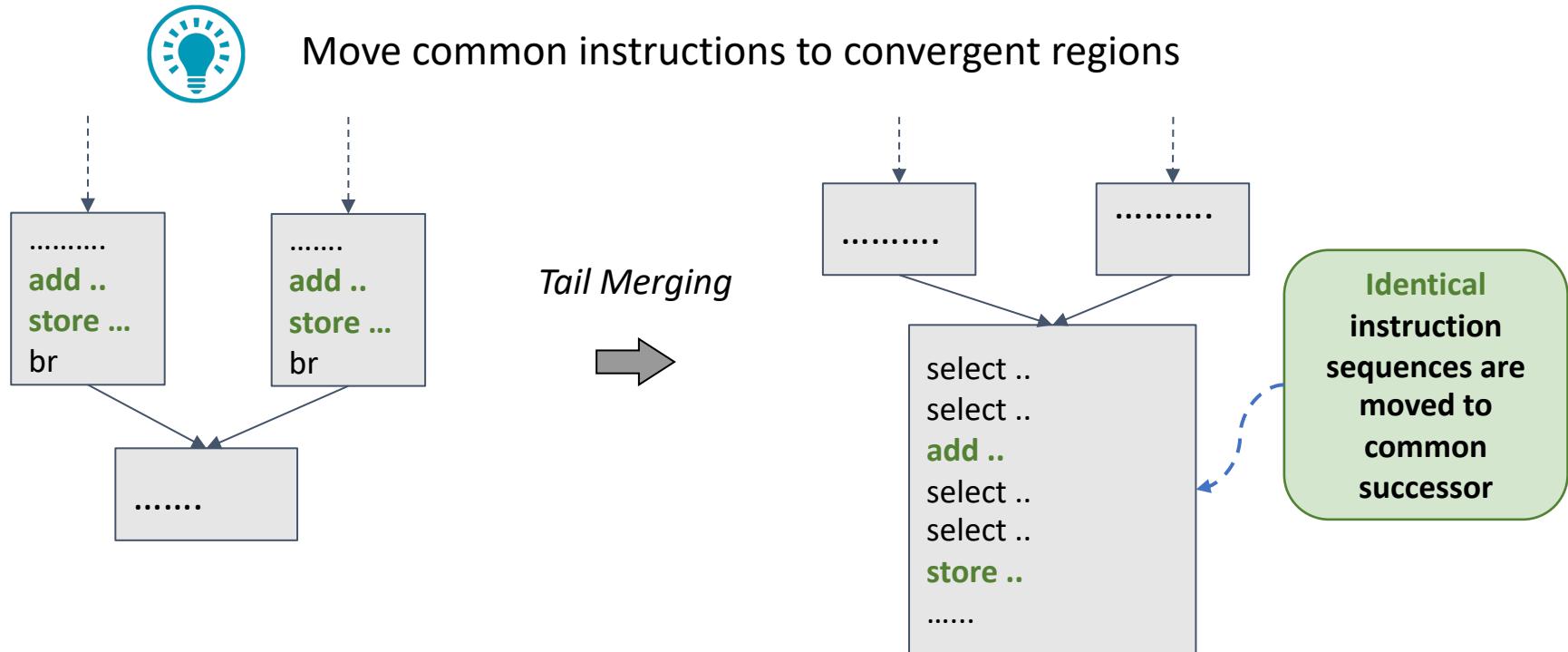
Control-Flow Divergence



Control-Flow Divergence

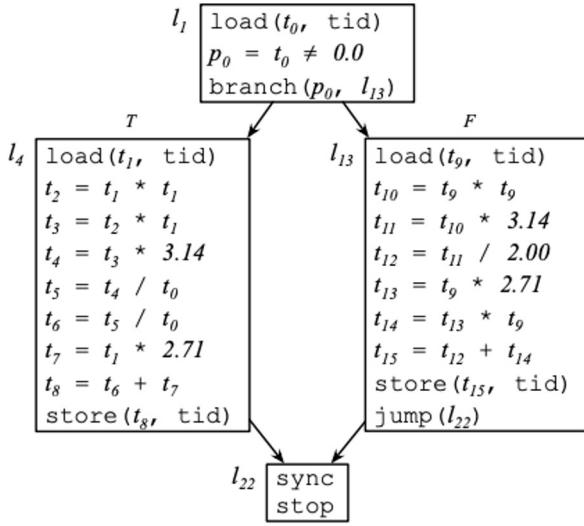


Existing Compiler-based Solutions



Existing Compiler based Solutions

(a)



*Branch Fusion
using
Instruction alignment*



(c)

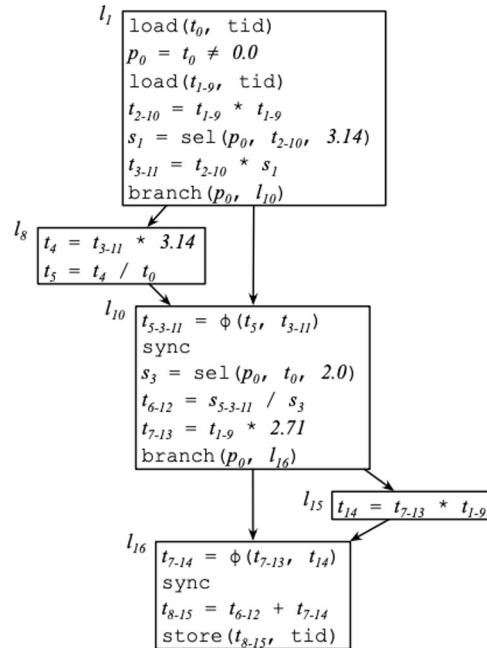
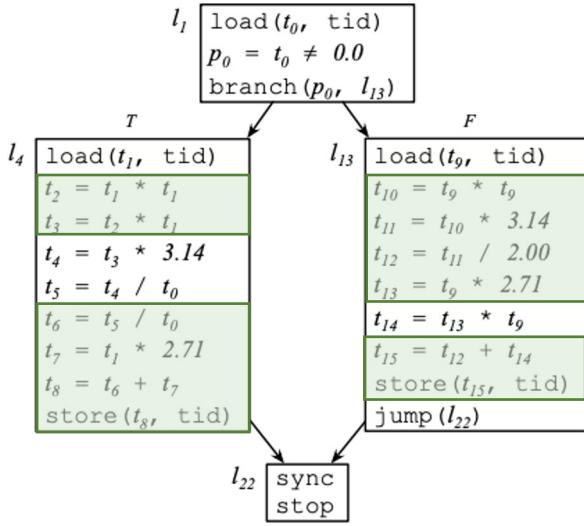


Image Source : B. Coutinho, D. Sampaio, F. M. Q. Pereira and W. Meira Jr., "Divergence Analysis and Optimizations," , PACT'11

Existing Compiler based Solutions

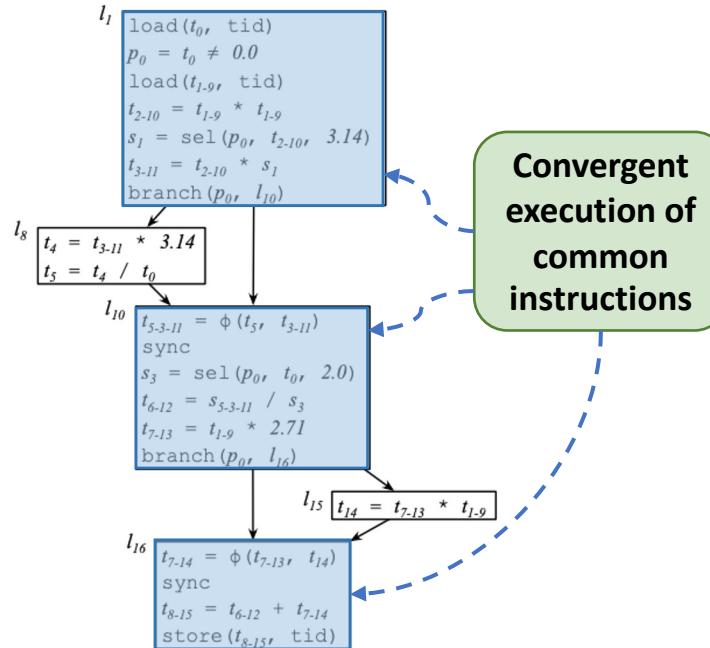
(a)



*Branch Fusion
using
Instruction alignment*

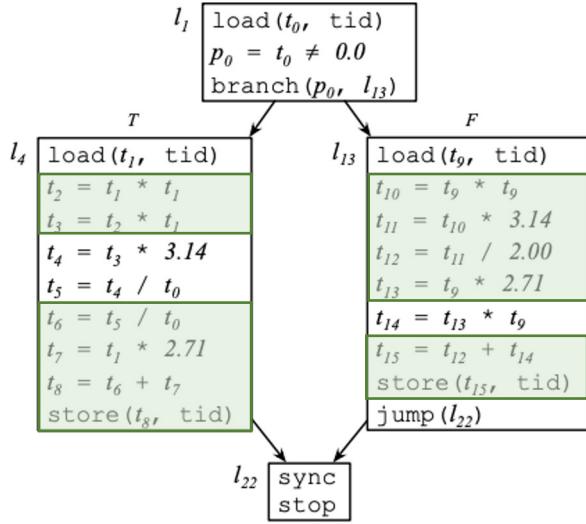


(c)



Existing Compiler based Solutions

(a)

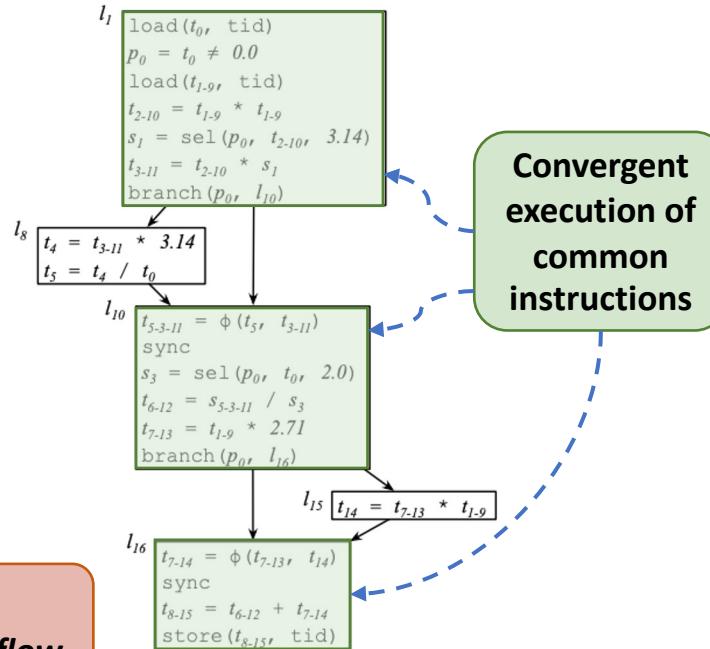


*Branch Fusion
using
Instruction alignment*



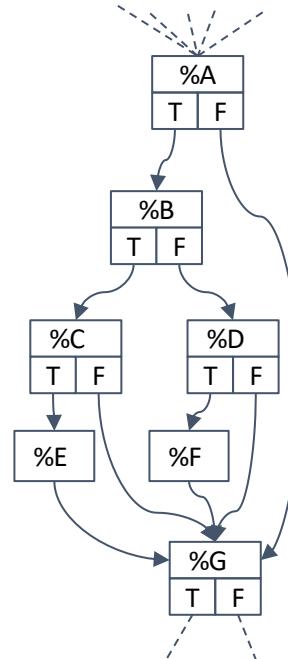
*Limited to
diamond-shaped control-flow*

(c)



Bitonic Sort

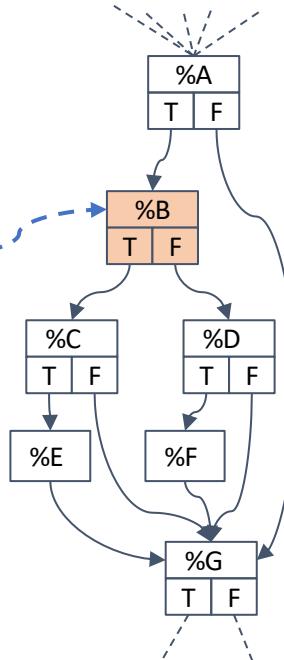
```
1 __global__ static void bitonicSort(int *values) {
2     // copy data from global memory to shared memory
3     __syncthreads();
4     for (unsigned int k = 2; k <= NUM; k *= 2) {
5         for (unsigned int j = k / 2; j > 0; j /= 2) {
6             unsigned int ixj = tid ^ j;
7             if (ixj > tid) {
8                 if ((tid & k) == 0) {
9                     if (shared[ixj] < shared[tid])
10                         swap(shared[tid], shared[ixj]);
11                 }
12             else {
13                 if (shared[ixj] > shared[tid])
14                     swap(shared[tid], shared[ixj]);
15             }
16         }
17         __syncthreads();
18     }
19 } // write data back to global memory
20 }
```



Bitonic Sort

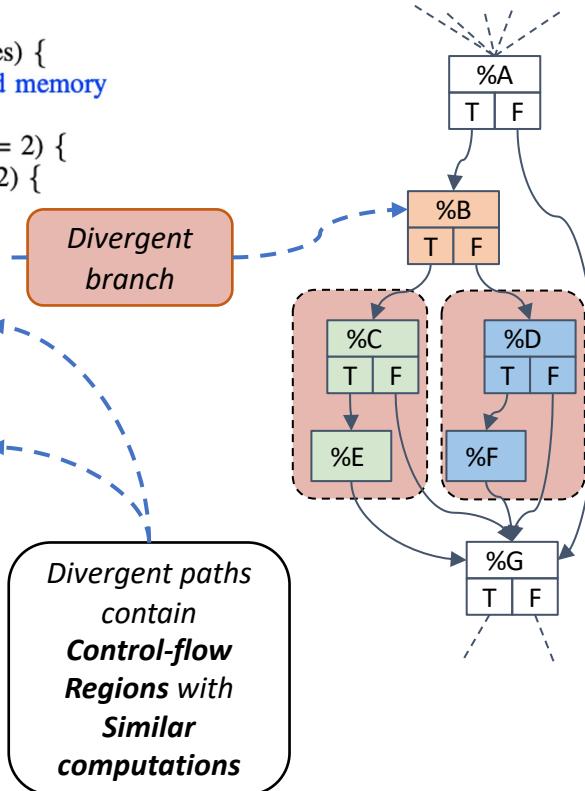
```
1 __global__ static void bitonicSort(int *values) {
2     // copy data from global memory to shared memory
3     __syncthreads();
4     for (unsigned int k = 2; k <= NUM; k *= 2) {
5         for (unsigned int j = k / 2; j > 0; j /= 2) {
6             unsigned int ixj = tid ^ j;
7             if (ixj > tid) {
8                 if ((tid & k) == 0) {
9                     if (shared[ixj] < shared[tid])
10                         swap(shared[tid], shared[ixj]);
11                 }
12             else {
13                 if (shared[ixj] > shared[tid])
14                     swap(shared[tid], shared[ixj]);
15             }
16         }
17     }
18     __syncthreads();
19 } // write data back to global memory
20 }
```

Divergent branch



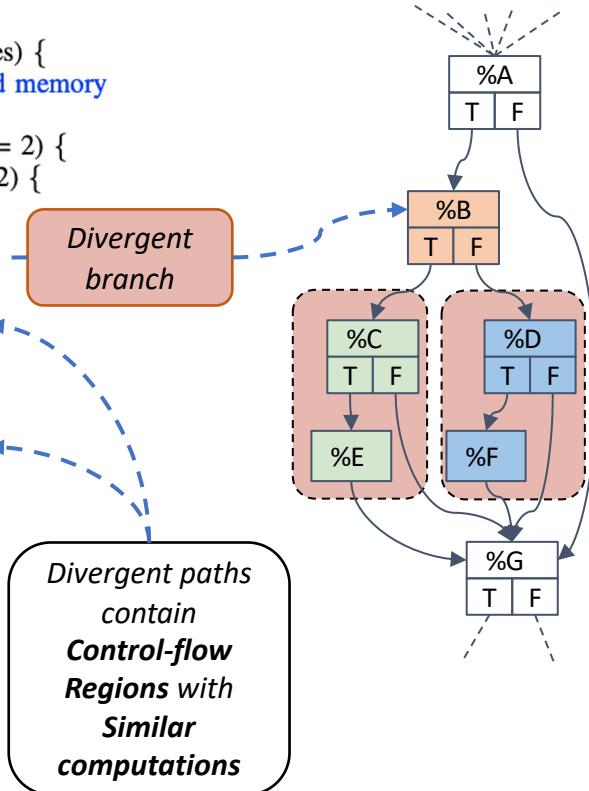
Bitonic Sort

```
1 __global__ static void bitonicSort(int *values) {
2     // copy data from global memory to shared memory
3     __syncthreads();
4     for (unsigned int k = 2; k <= NUM; k *= 2) {
5         for (unsigned int j = k / 2; j > 0; j /= 2) {
6             unsigned int ixj = tid ^ j;
7             if (ixj > tid) {
8                 if ((tid & k) == 0) {
9                     if (shared[ixj] < shared[tid])
10                         swap(shared[tid], shared[ixj]);
11                 }
12             } else {
13                 if (shared[ixj] > shared[tid])
14                     swap(shared[tid], shared[ixj]);
15             }
16         }
17     }
18     __syncthreads();
19 } // write data back to global memory
20 }
```



Bitonic Sort

```
1 __global__ static void bitonicSort(int *values) {
2     // copy data from global memory to shared memory
3     __syncthreads();
4     for (unsigned int k = 2; k <= NUM; k *= 2) {
5         for (unsigned int j = k / 2; j > 0; j /= 2) {
6             unsigned int ixj = tid ^ j;
7             if (ixj > tid) {
8                 if ((tid & k) == 0) {
9                     if (shared[ixj] < shared[tid])
10                         swap(shared[tid], shared[ixj]);
11                 }
12             } else {
13                 if (shared[ixj] > shared[tid])
14                     swap(shared[tid], shared[ixj]);
15             }
16         }
17         __syncthreads();
18     }
19 } // write data back to global memory
```

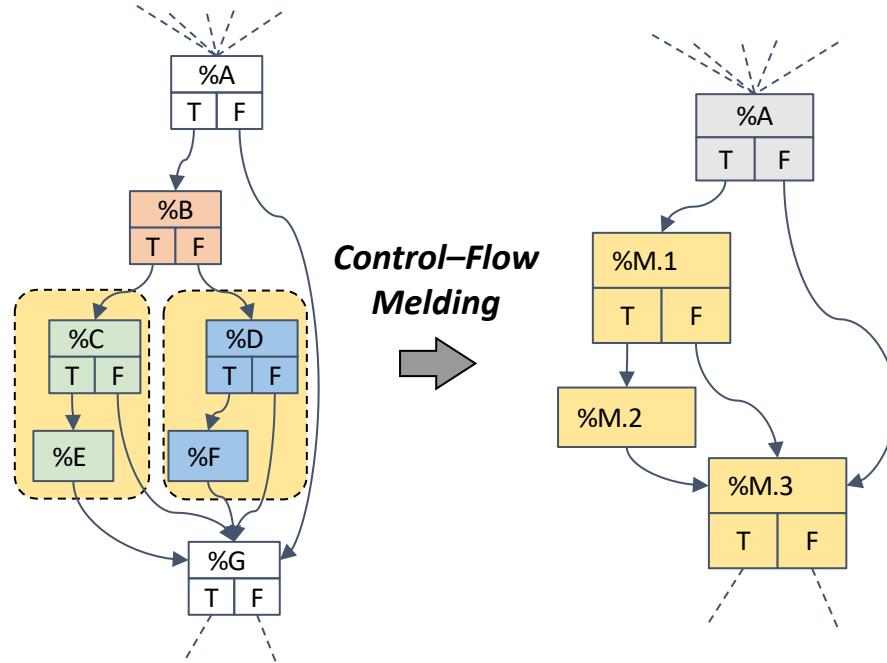


*Tail Merging or
Branch Fusion
cannot merge
control-flow regions*



Control-Flow Melding

```
1 __global__ static void bitonicSort(int *values) {
2     // copy data from global memory to shared memory
3     __syncthreads();
4     for (unsigned int k = 2; k <= NUM; k *= 2) {
5         for (unsigned int j = k / 2; j > 0; j /= 2) {
6             unsigned int ixj = tid ^ j;
7             if (ixj > tid) {
8                 if ((tid & k) == 0) {
9                     if (shared[ixj] < shared[tid])
10                         swap(shared[tid], shared[ixj]);
11                 }
12             } else {
13                 if (shared[ixj] > shared[tid])
14                     swap(shared[tid], shared[ixj]);
15             }
16             __syncthreads();
17         }
18     }
19 } // write data back to global memory
20 }
```

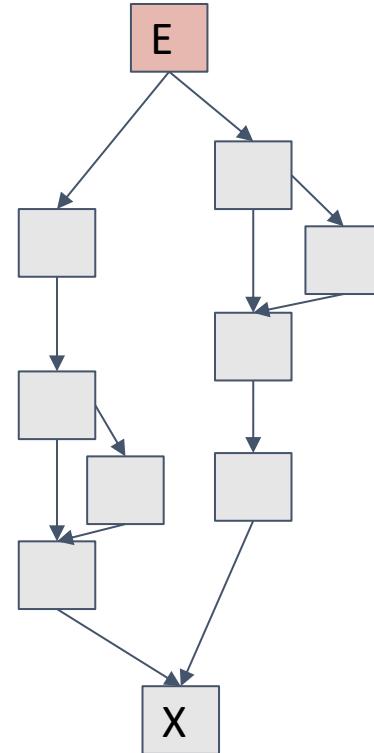


Contributions

- Divergence-Aware-Region-Melder (DARM), a realization of Control-Flow-Melding that can find and meld similar control-flow regions to reduce divergence
- Implementation of DARM in LLVM
- Evaluation of DARM on synthetic and real-world benchmarks showing its effectiveness

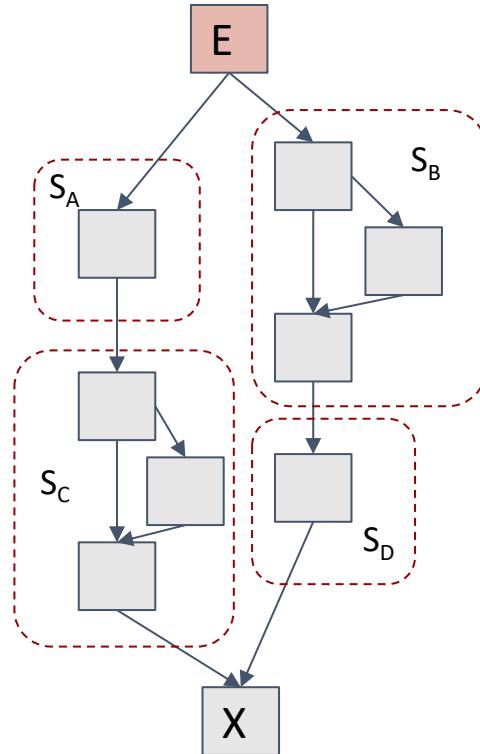
Detecting Divergent Regions

- **Divergent Region**
 - E has a divergent branch
 - E 's two successors do not post-dominate each other



Single-Entry Single-Exit (SESE) Subgraphs

- Split the region into SESE subgraphs
- SESE subgraph
 - SESE region (e.g. S_B, S_C)
 - Single basic block with single successor and single predecessor (e.g. S_A, S_D)



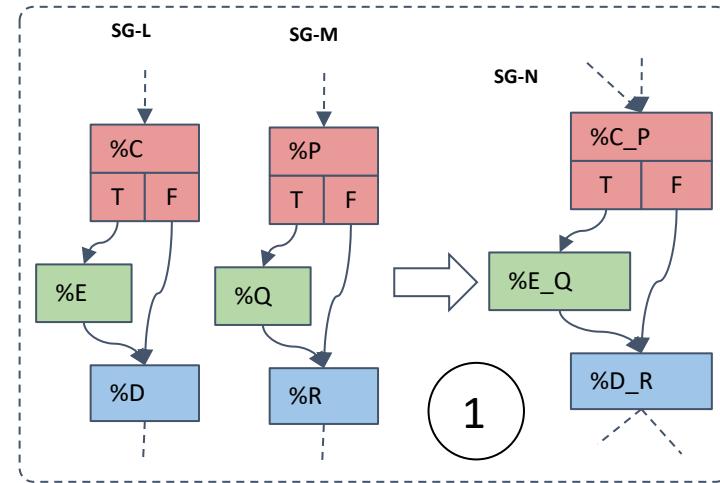
Meldable Subgraphs

Two subgraphs are *meldable* if they fall into one of following 3 categories

1

Region - Region

- Contains more than one basic block
- Isomorphic



Meldable Subgraphs

Two subgraphs are *meldable* if they fall into one of following 3 categories

1

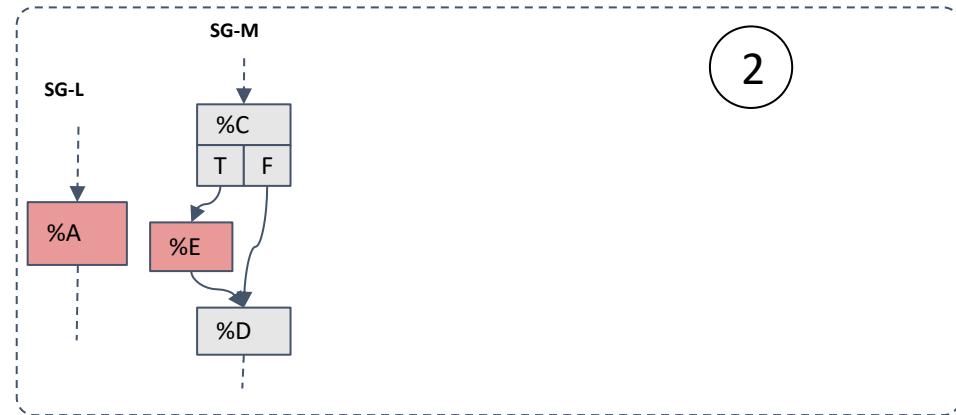
Region - Region

- Contains more than one basic block
- Isomorphic

2

Basic block - Region

- Single basic block expanded to match a region (aka *Region Replication*)



Meldable Subgraphs

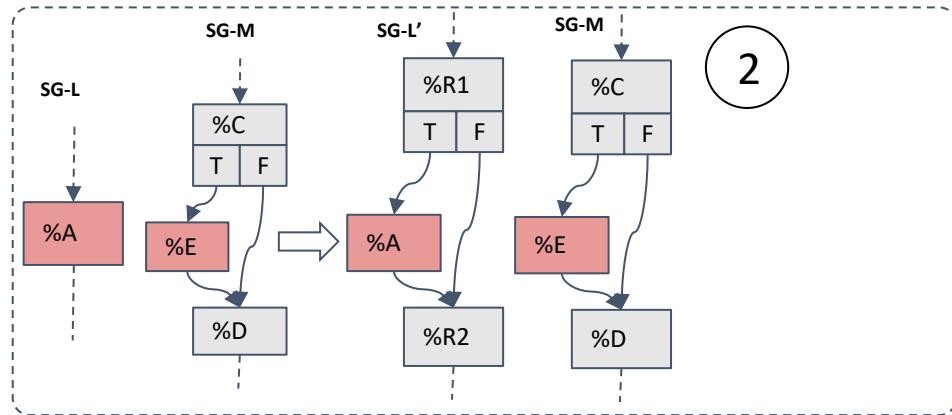
Two subgraphs are *meldable* if they fall into one of following 3 categories

1 Region - Region

- Contains more than one basic block
- Isomorphic

2 Basic block - Region

- Single basic block expanded to match a region (aka *Region Replication*)



Meldable Subgraphs

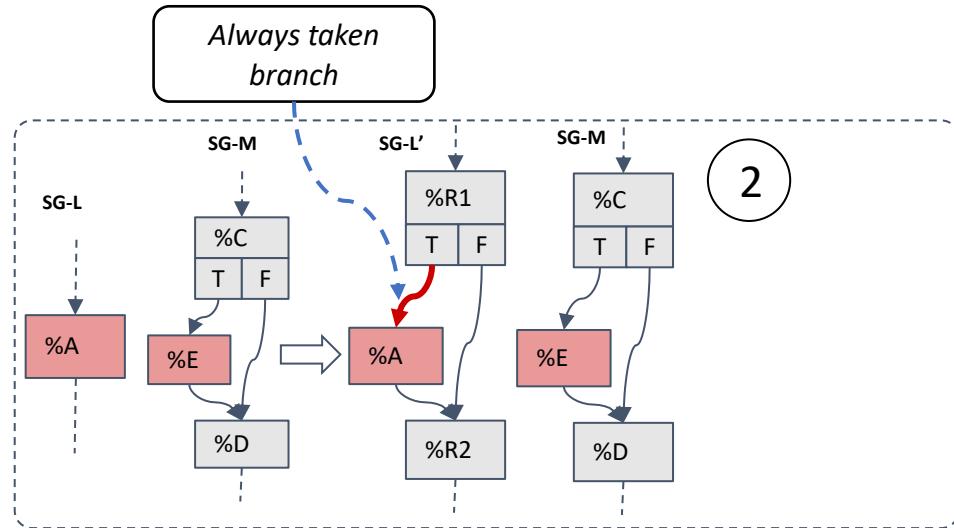
Two subgraphs are *meldable* if they fall into one of following 3 categories

1 Region - Region

- Contains more than one basic block
- Isomorphic

2 Basic block - Region

- Single basic block expanded to match a region (aka *Region Replication*)



Meldable Subgraphs

Two subgraphs are *meldable* if they fall into one of following 3 categories

1

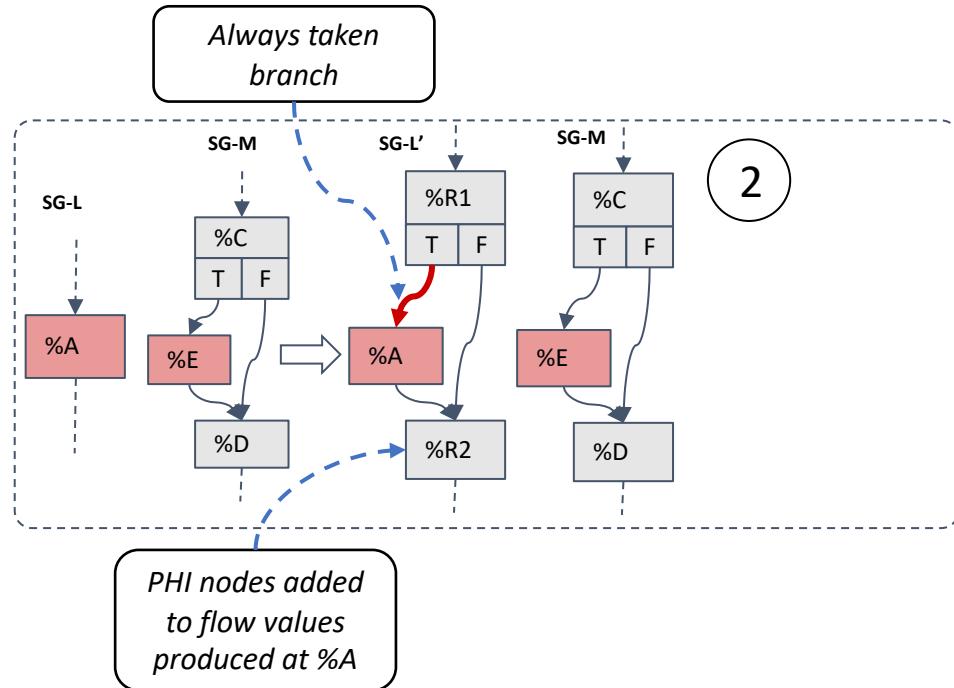
Region - Region

- Contains more than one basic block
- Isomorphic

2

Basic block - Region

- Single basic block expanded to match a region (aka *Region Replication*)



Meldable Subgraphs

Two subgraphs are *meldable* if they fall into one of following 3 categories

1

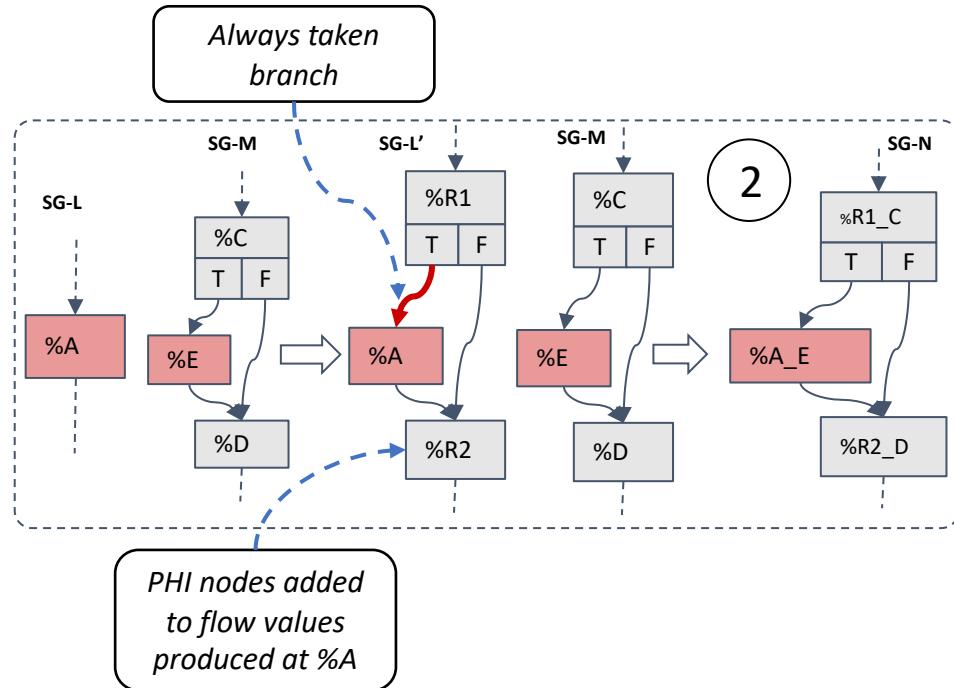
Region - Region

- Contains more than one basic block
- Isomorphic

2

Basic block - Region

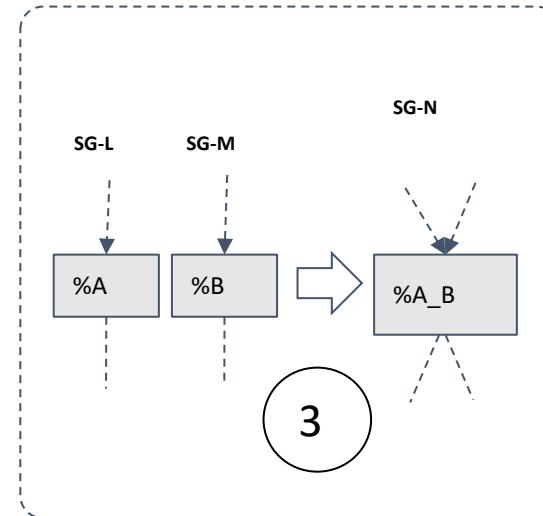
- Single basic block expanded to match a region (aka *Region Replication*)



Meldable Subgraphs

Two subgraphs are ***meldable*** if they fall into one of following 3 categories

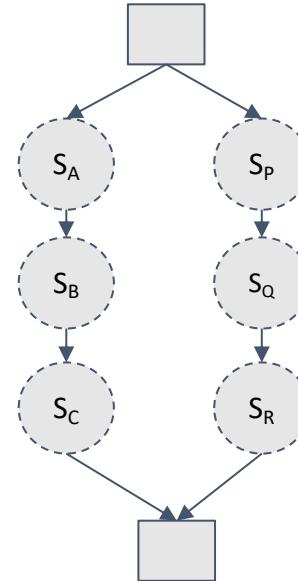
- 1 Region - Region
 - Contains more than one basic block
 - Isomorphic
- 2 Basic block - Region
 - Single basic block expanded to match a region (aka *Region Replication*)
- 3 Basic block – Basic block



Profitable Subgraph Alignment

- A divergent region → many SESE subgraphs

How to decide which subgraphs to meld?

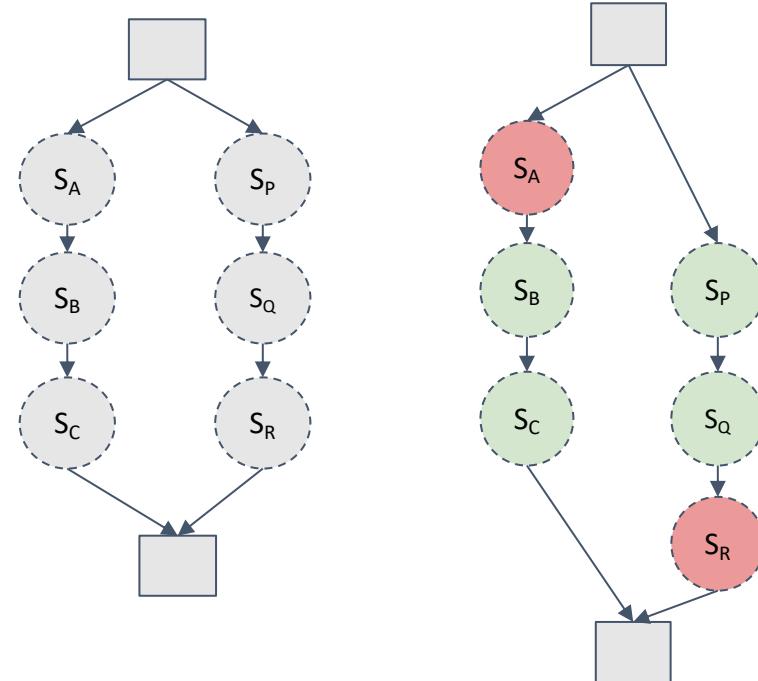


Profitable Subgraph Alignment

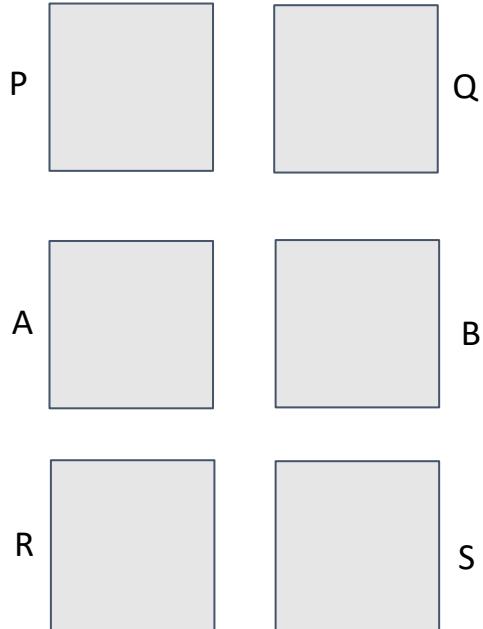
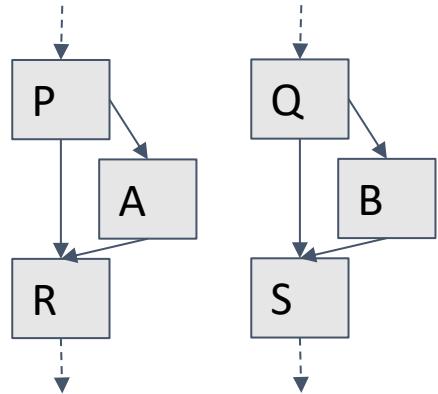
- A divergent region → many SESE subgraphs

How to decide which subgraphs to meld?

- Subgraphs are aligned based on their **Melding Profitability**
- **Melding Profitability** : computed based on *instruction frequencies*

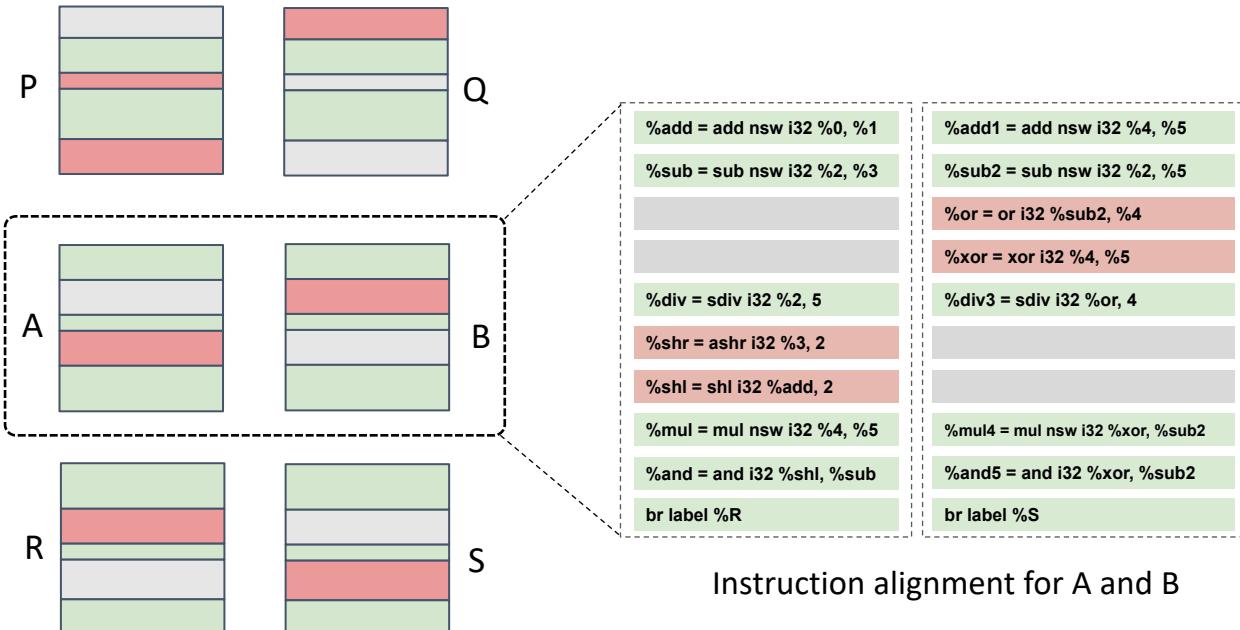
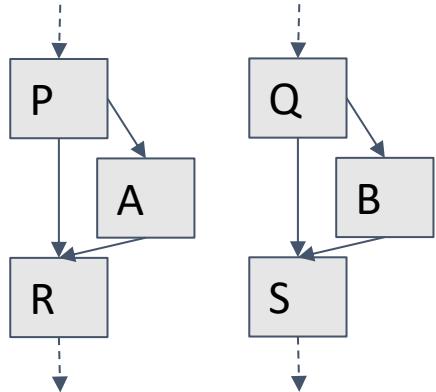


Subgraph Instruction Alignment

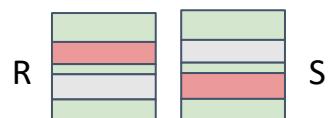
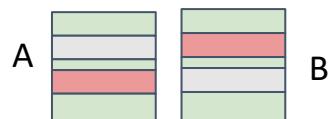
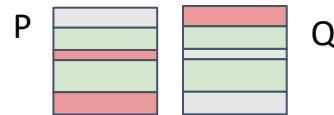
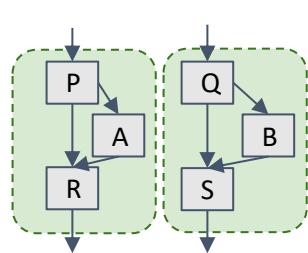


Pre-order Linearization

Subgraph Instruction Alignment

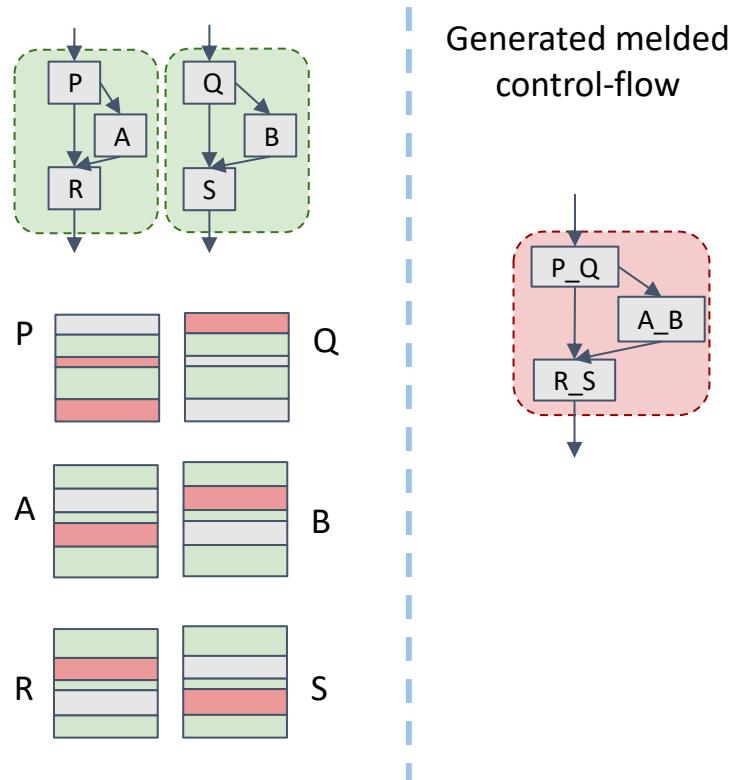


DARM Code Generation

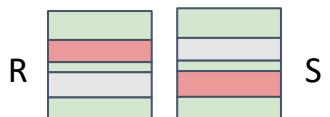
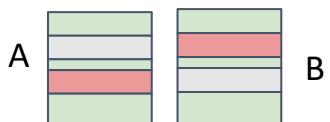
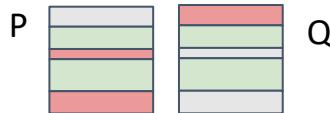
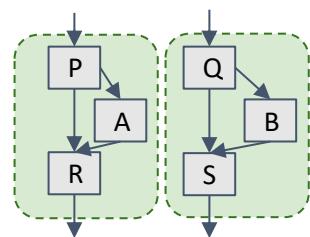


36

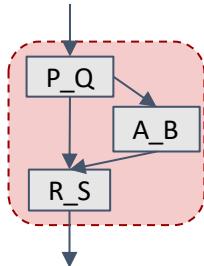
DARM Code Generation



DARM 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

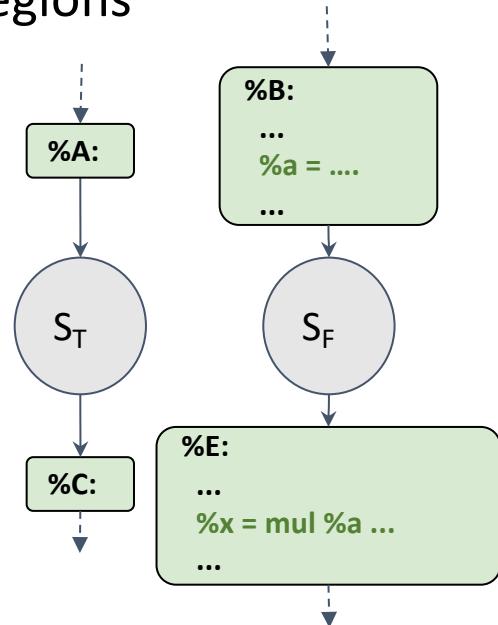
%sel1 = select i1 %cmp, i32 %0, i32 %4

%sel2 = select i1 %cmp, i32 %1, i32 %5

%6 = add nsw i32 %sel1, %sel2

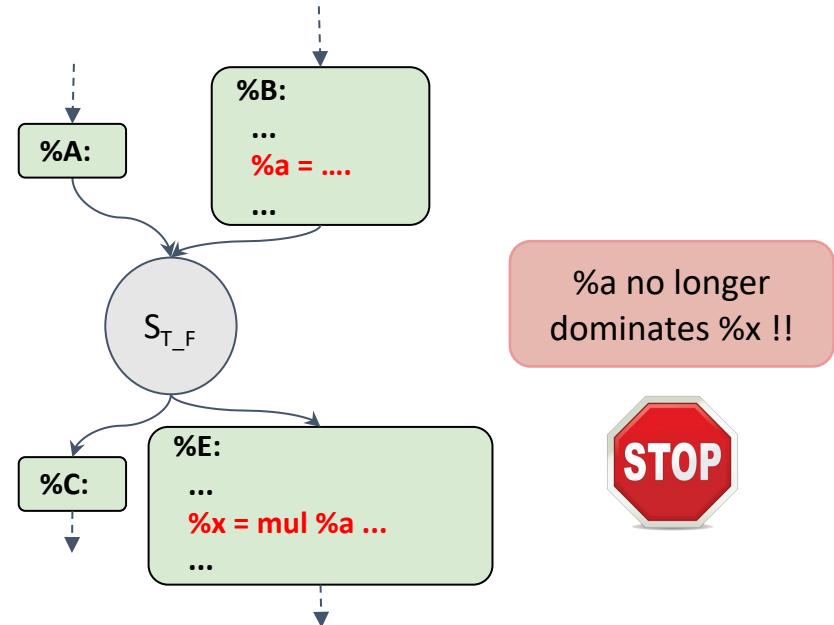
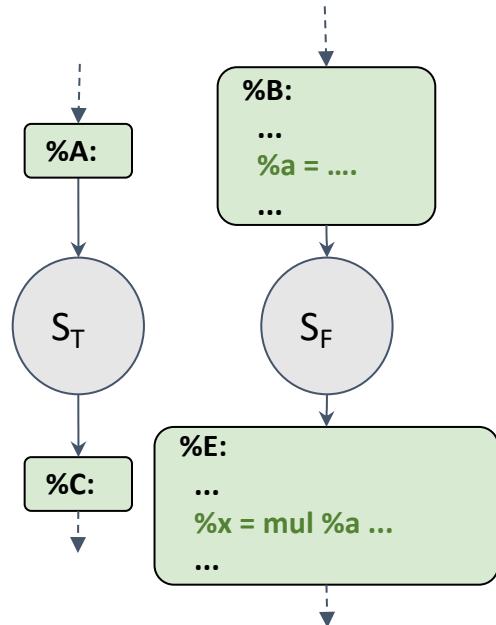
Ensuring Correctness

- Melding can break the use-def chains that go through the melded regions



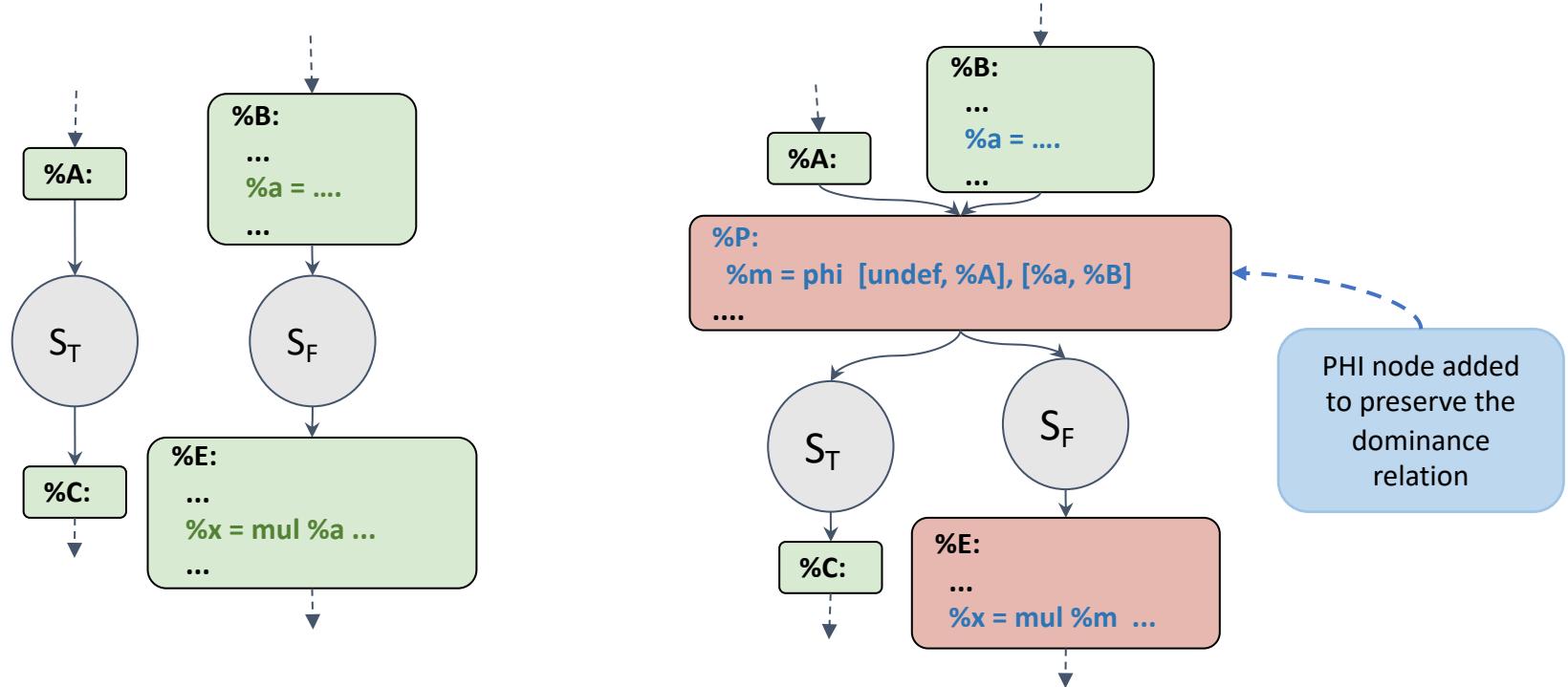
Ensuring Correctness

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



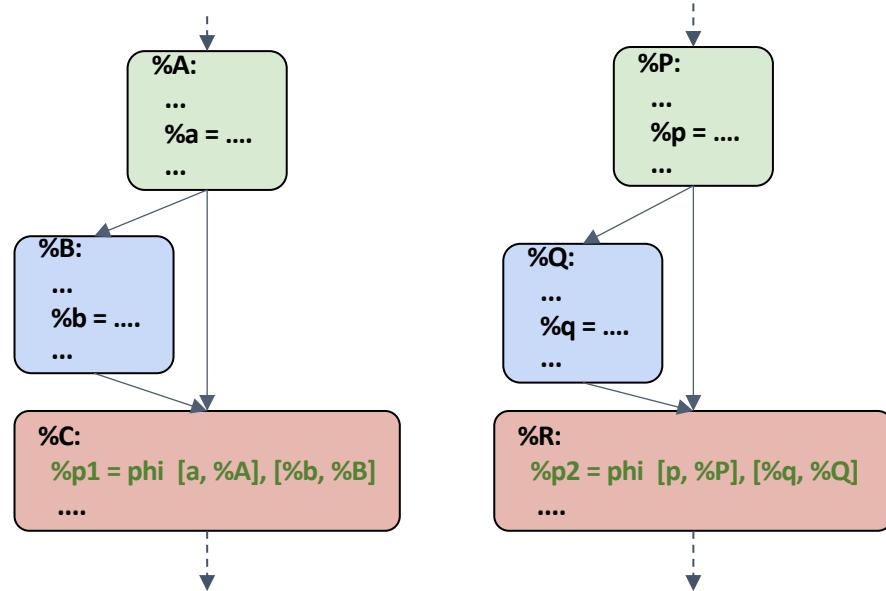
Ensuring Correctness

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



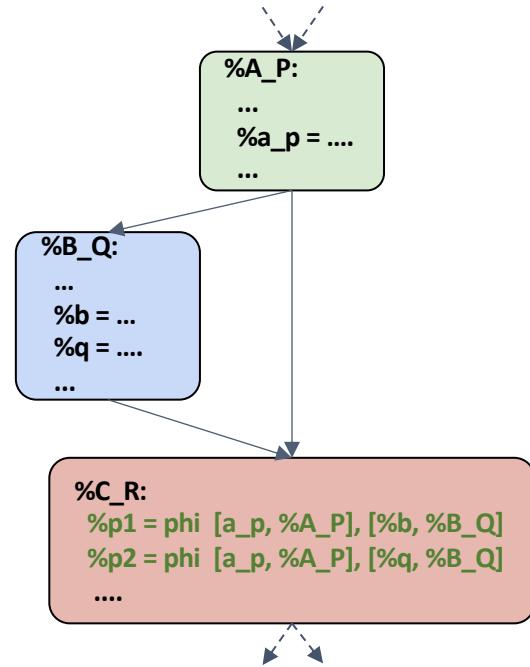
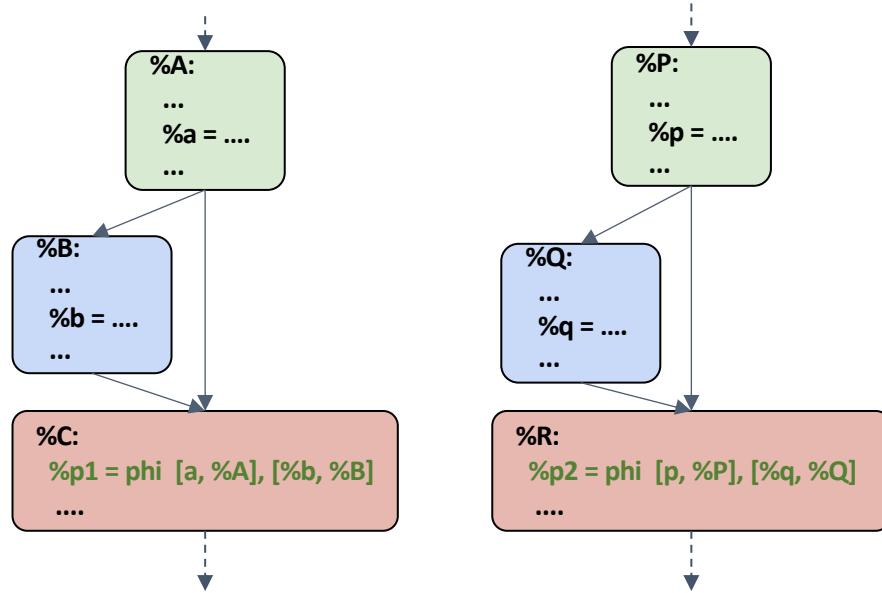
Ensuring Correctness

- Melding PHI nodes



Ensuring Correctness

- Melding PHI instructions



Ensuring Correctness

- Melding unaligned instructions

%add = add nsw i32 %0, %1	%add1 = add nsw i32 %4, %5
%sub = sub nsw i32 %2, %3	%sub2 = sub nsw i32 %2, %5
	%a = or i32 %sub2, %4
	%b = xor i32 %4, %5
%div = sdiv i32 %2, 5	%div3 = sdiv i32 %a, 4

Full-predication of **unaligned instructions** can be problematic

- False positives in divergence analysis
- Instructions with side effects



Ensuring Correctness

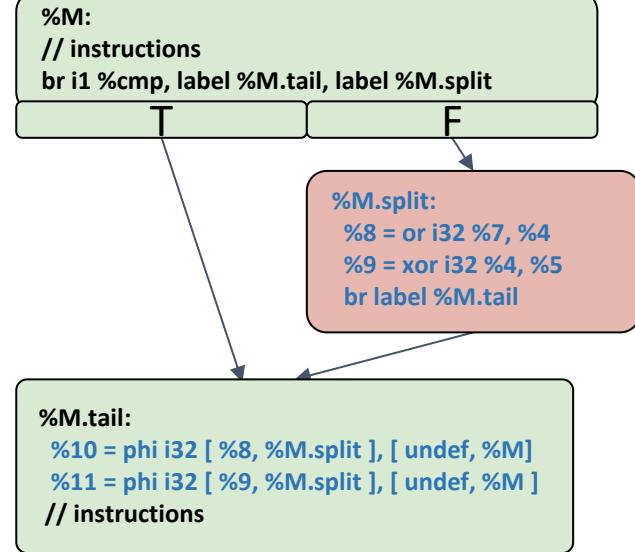
- Unpredication

```
%add = add nsw i32 %0, %1
%sub = sub nsw i32 %2, %3
%
%
%
%div = sdiv i32 %2, 5
```

```
%add1 = add nsw i32 %4, %5
%sub2 = sub nsw i32 %2, %5
%
%
%
%a = or i32 %sub2, %4
%b = xor i32 %4, %5
%
%
%
%div3 = sdiv i32 %a, 4
```



Execute unaligned instructions conditionally



Ensuring Correctness

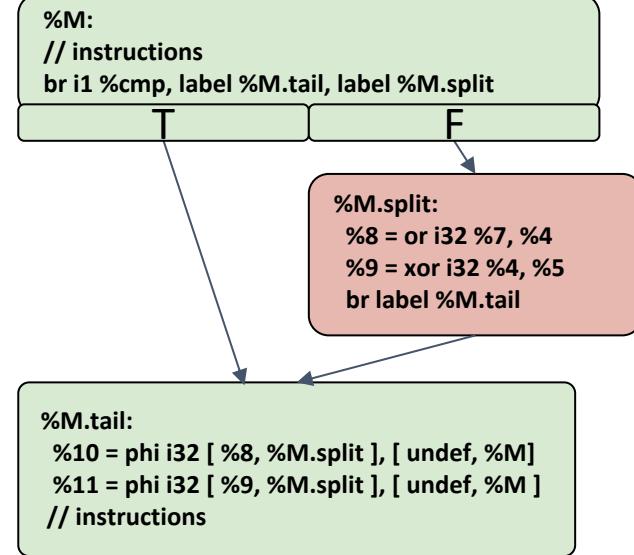
- Melding unaligned instructions

```
%add = add nsw i32 %0, %1
%sub = sub nsw i32 %2, %3
%div = sdiv i32 %2, 5
%add1 = add nsw i32 %4, %5
%sub2 = sub nsw i32 %2, %5
%a = or i32 %sub2, %4
%b = xor i32 %4, %5
%div3 = sdiv i32 %a, 4
```

*More Details in
the paper!*

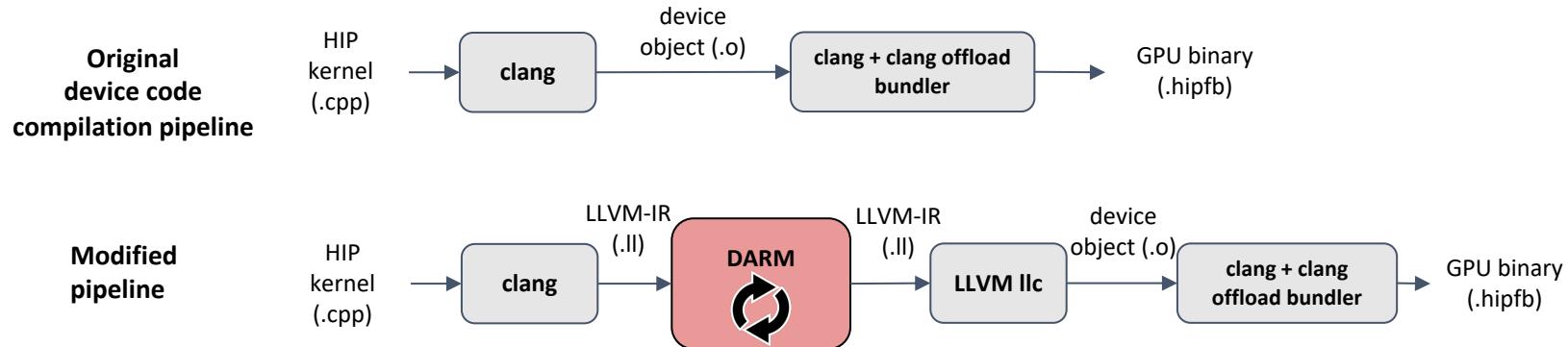


Execute unaligned instructions
conditionally



Implementation

- LLVM pass integrated with LLVM *opt*
- Uses LLVM's built-in Divergence Analysis
- Currently Targeting AMD GPUs with *ROCM HIPCC* compiler



Implementation

- LLVM pass integrated with LLVM *opt*
- Uses LLVM's built-in Divergence Analysis
- Currently Targeting AMD GPUs with *ROCm HIPCC* compiler

Evaluation Setup

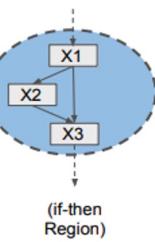
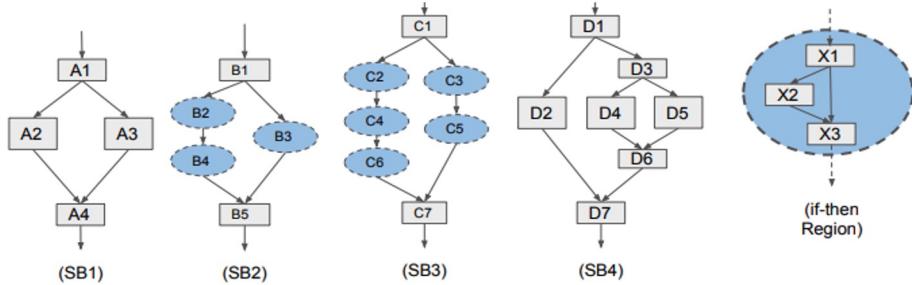
Hardware : AMD Radeon Pro Vega 20 GPU + AMD Ryzen CPU

Baseline : Full-optimizations (-O3)

Branch Fusion : Implemented on top of DARM

Benchmarks

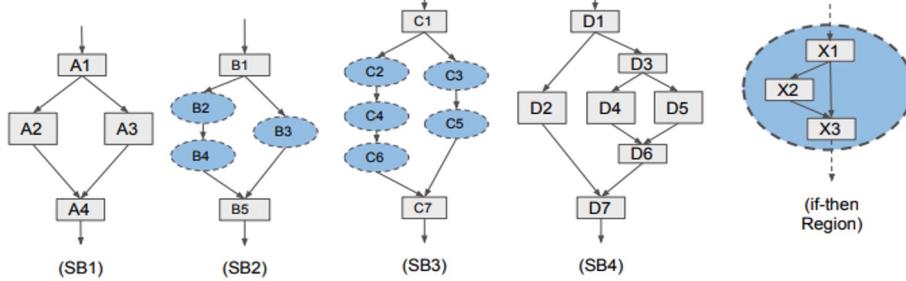
Synthetic Benchmarks



Access DARM's
generality using a
variety of divergent
control-flow patterns

Benchmarks

Synthetic Benchmarks



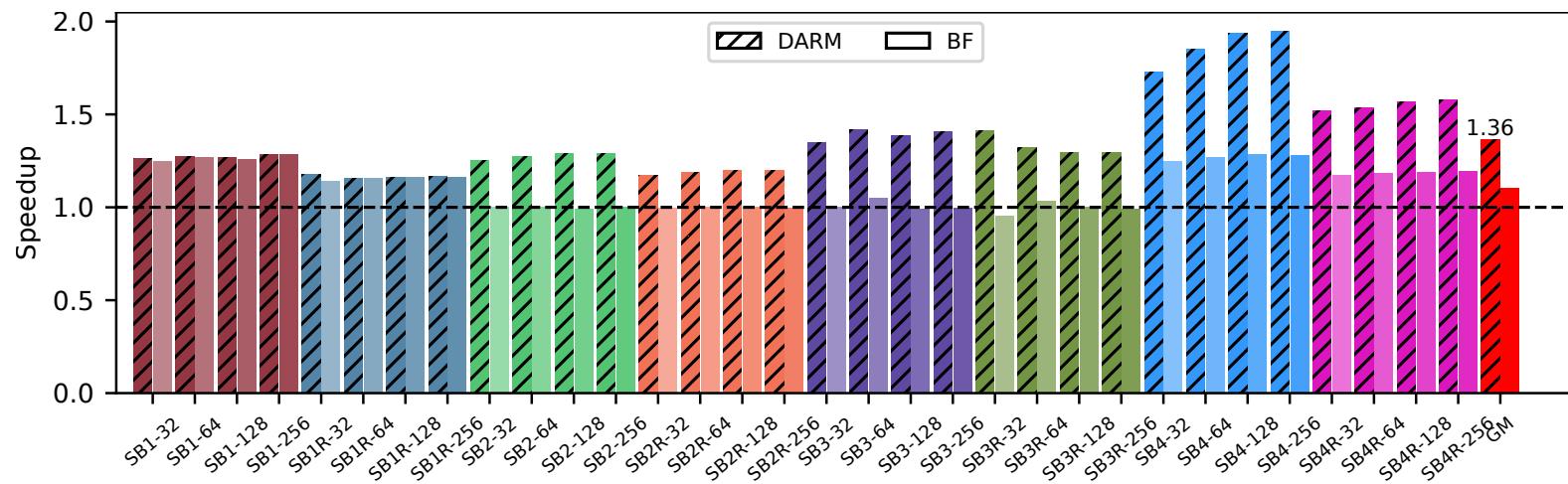
Access DARM's generality using a variety of divergent control-flow patterns

Real-World 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)

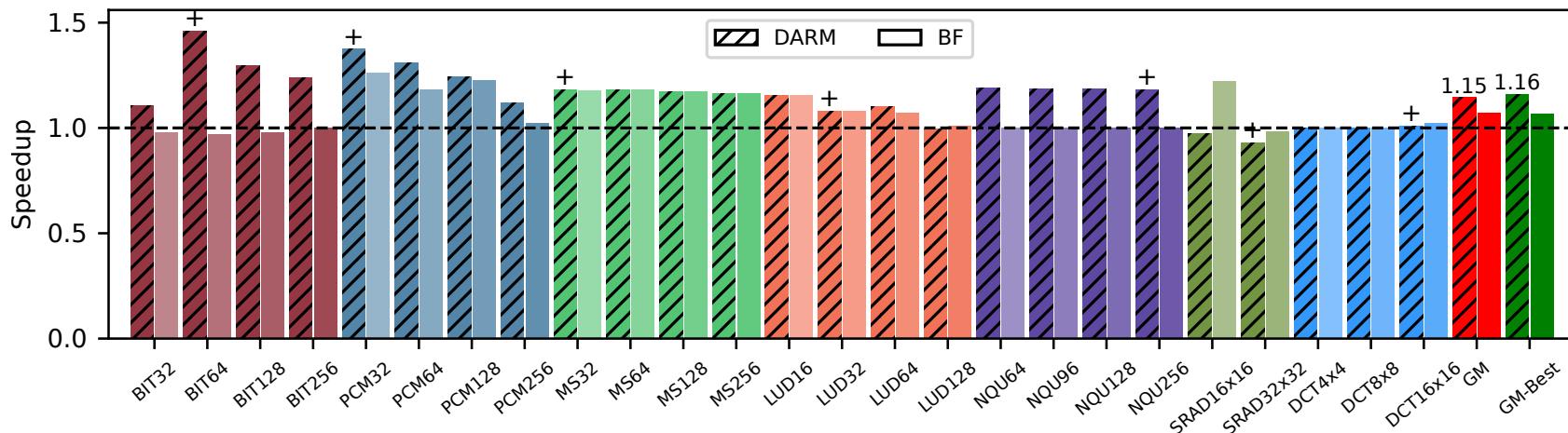
Already heavily optimized

Synthetic Benchmarks Performance

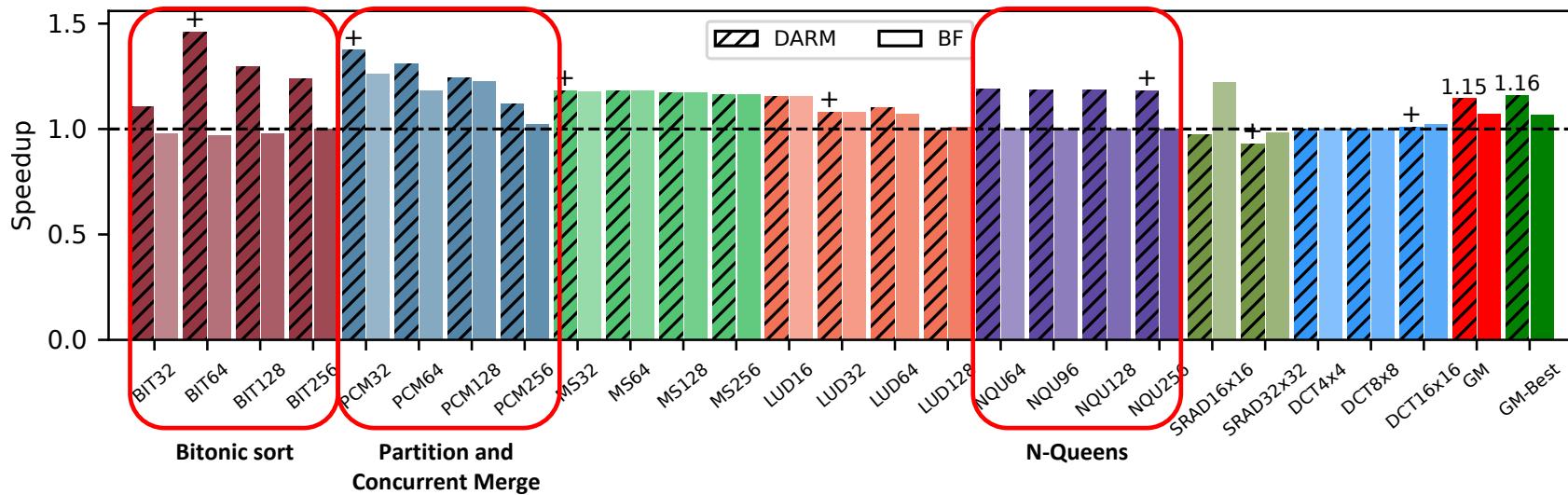


DARM is more general than *Branch Fusion* and can reduce control-flow divergence in a variety of cases

Real-World Benchmarks Performance

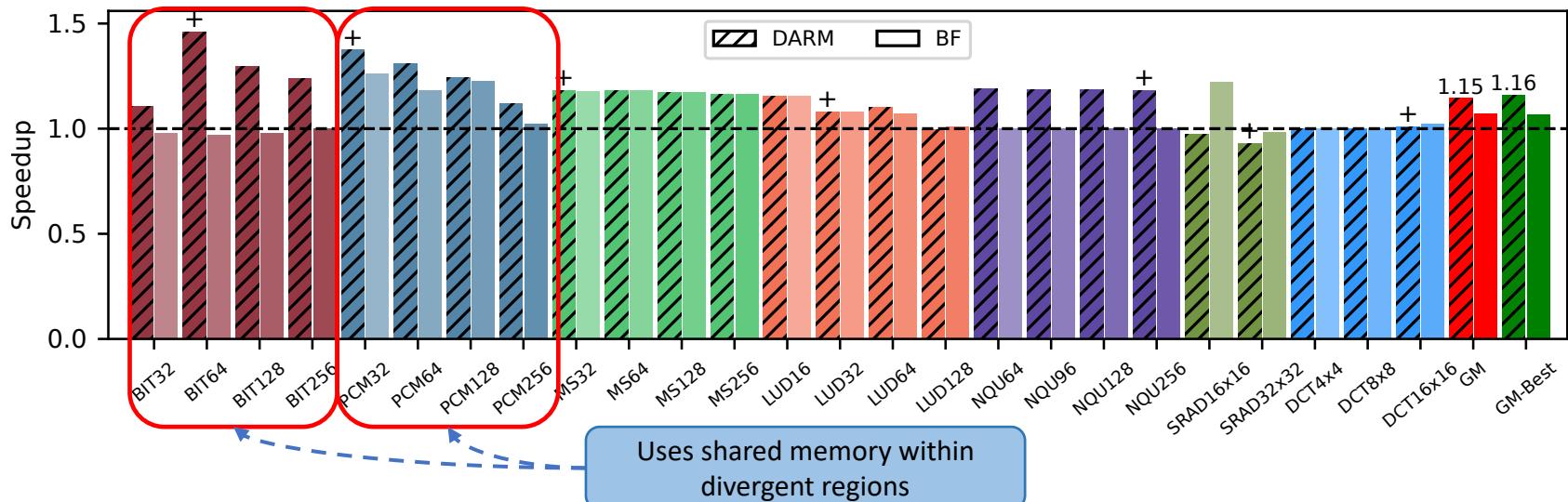


Real-World Benchmarks Performance



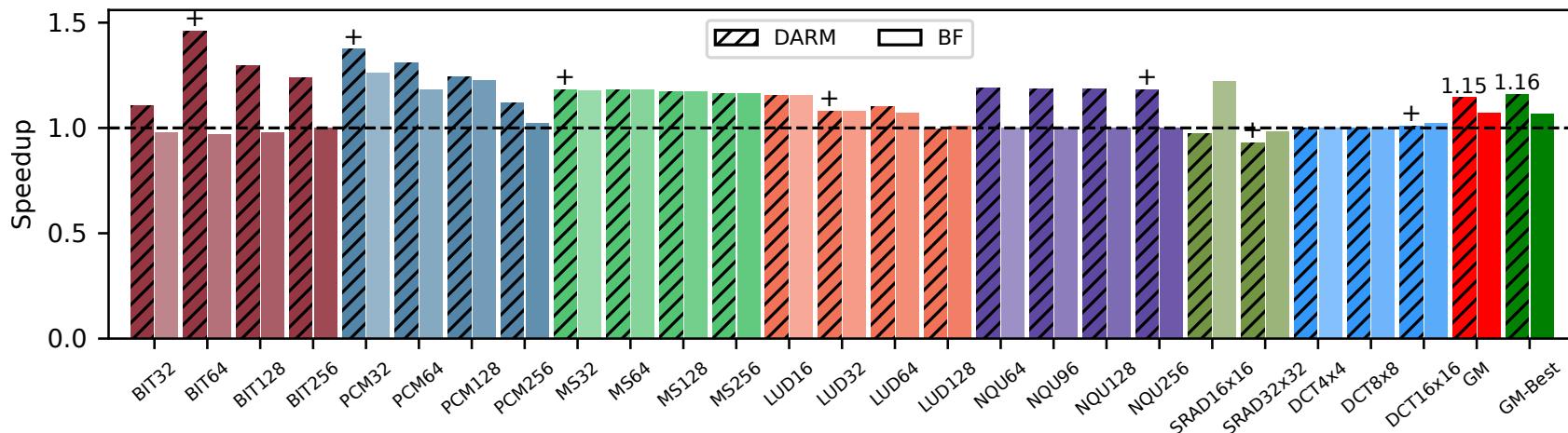
Only DARM can fully meld the control-flow structures present in
BIT, PCM and NQU

Real-World Benchmarks Performance



Melding shared memory instructions is important for achieving good performance

Real-World Benchmarks Performance



DARM either matches or improves the performance of Branch Fusion or Baseline (in most cases)

Compile-time Overhead

Benchmark	O3 (seconds)	DARM (seconds)	Normalized
BIT	0.4804	0.5018	1.0444
PCM	0.5690	0.5942	1.0443
MS	0.8037	0.8064	1.0035
LUD	0.5993	0.6294	1.0502
NQU	0.4687	0.4738	1.0109
SRAD	0.4999	0.5121	1.0244
DCT	0.4398	0.4439	1.0093

Compile-time overhead of DARM is not significant

Compile-time Overhead

Benchmark	O3 (seconds)	DARM (seconds)	Normalized
BIT	0.4804	0.5018	1.0444
PCM	0.5690	0.5942	1.0443
MS	0.8037	0.8064	1.0035
LUD	0.5993	0.6294	1.0502
NQU	0.4687	0.4738	1.0109
SRAD	0.4999	0.5121	1.0244
DCT	0.4398	0.4439	1.0093

Checkout our paper for more detailed evaluation of DARM
including ALU utilization, memory instruction counters

Summary

- Control-flow divergence can be a significant performance bottleneck in GPU programs
- Existing compiler optimizations are not general enough
- We propose **DARM**, a general framework for reducing control-flow divergence in the presence of arbitrary control-flow
- DARM shows good performance on real-world benchmarks
- **Future work** : Applicability of DARM for code size reduction and accelerating program testing

Contact :

cgusthin@purdue.edu

Code :

github.com/charitha22/cgo22ae-darm-code

Benchmarks :

github.com/charitha22/cgo22ae-darm-benchmarks



*This presentation and recording belong to the authors. No distribution
is allowed without the authors' permission*

DARM: Control-Flow Melding for SIMT Thread Divergence Reduction

Charitha Saumya, Kirshanthan Sundararajah, Milind Kulkarni



CGO '22

April 2nd – 6th 2022