Uniformity Analysis for Irreducible CFGs

Sameer Sahasrabuddhe*
Nicolai Hähnle

AMD
together we advance_

Motivation

- Support for irreducible CFGs
  - Existing Divergence Analysis conservatively assumes that all values are non-uniform.
- Implementation for Machine IR
  - Existing DA is implemented only on LLVM IR.
  - Uniformity Analysis is a template that works for LLVM IR and Machine IR.

Overview

- Uniformity of values is closely related to thread convergence.
- Start with a definition of convergent execution:
  - **Static and Dynamic Instances** of Operations.
  - Dynamic instances related by `converged-with`.
  - **Maximal Convergence:** A `converged-with` relation suitable for known use-cases.
- `m-converged` Static Instances
  - Static property suitable for irreducible CFGs.
  - Derived from the `converged-with` relation over dynamic instances.
- **Uniformity** defined using `m-converged` Static Instances.

References

The Uniformity Analysis extends the following work by introducing a conservative treatment of irreducible control flow graphs.

- Julian Rosemann, Simon Moll, and Sebastian Hack, “An Abstract Interpretation for SPMD Divergence on Reducible Control Flow Graphs” in Proc. ACM Program. Lang. 5, POPL, Article 31 (January 2021), 35 pages.

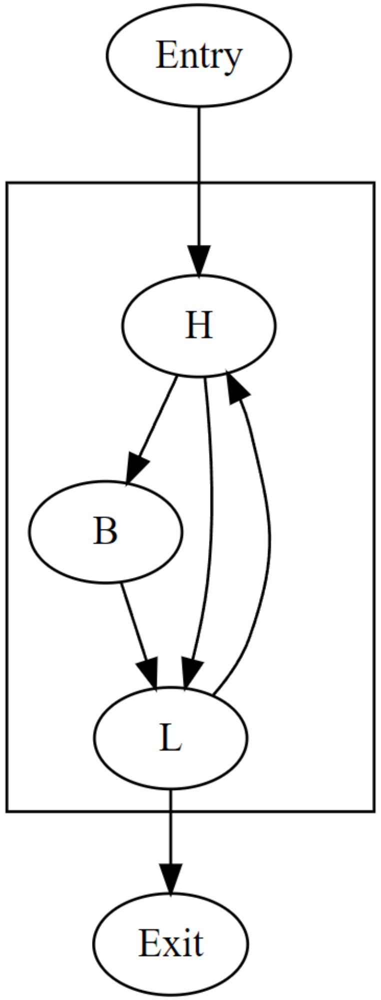
  https://doi.org/10.1145/3434312

Convergence and Uniformity

- Conventional picture:
  - Threads are **converged** until they **diverge** at a *divergent* branch.
  - Diverged threads eventually **reconverge** at some common program point.
  - **Convergent operations** require certain threads to execute them convergently.
- Convergently executed operations produce uniform values (*conditions apply*).
- A value computed by different threads is ***uniform*** if it is the same across those threads.
  - The value is ***divergent*** otherwise.
- A branch is **uniform** or **divergent** if its condition is uniform or divergent, respectively.

Object	**Can be …**	
Thread	Converged	Diverged
Communication	Convergent	Independent
Value	Uniform	Divergent
Branch	Uniform	Divergent

Dynamic Instances



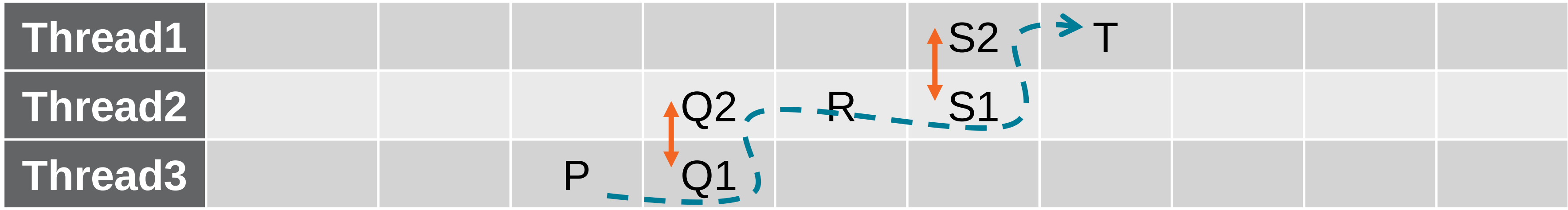
- **Static instance:** Each occurrence of an instruction in the program source.
    - E.g.: The nodes H, B, L, etc in the adjoining CFG.
- **Dynamic instance:** Each execution of a static instance by a thread.
    - E.g.: The entries H1, B1, H2, etc in the table below

Thread1	Entry1	H1	B1	L1	H3		L3	H5	L5	Exit1
Thread2	Entry2	H2		L2	H4	B2	L4			Exit2

Convention: Dynamic instances are listed in the same column if and only if they are `converged`.

Convergence = { `converged-with` and `convergence-before` }

- **converged-with**
  - Relates dynamic instances of the same static instance produced by different threads.
  - Transitive symmetric relation.
  - **No single definition**: Choose an instance that reflects the execution on the target.
- **convergence-before**
  - Produced by converged dynamic instances.
  - Relates other dynamic instances in the corresponding threads.
  - Transitive strict partial order.



Thread1						S2	T			
Thread2				Q2	R	S1				
Thread3			P	Q1						

- Converged:
  - Q1 and Q2.
  - S1 and S2.

- Convergence order:
  - `P  -> Q2`
  - `Q1 -> R`
  - `P  -> T`

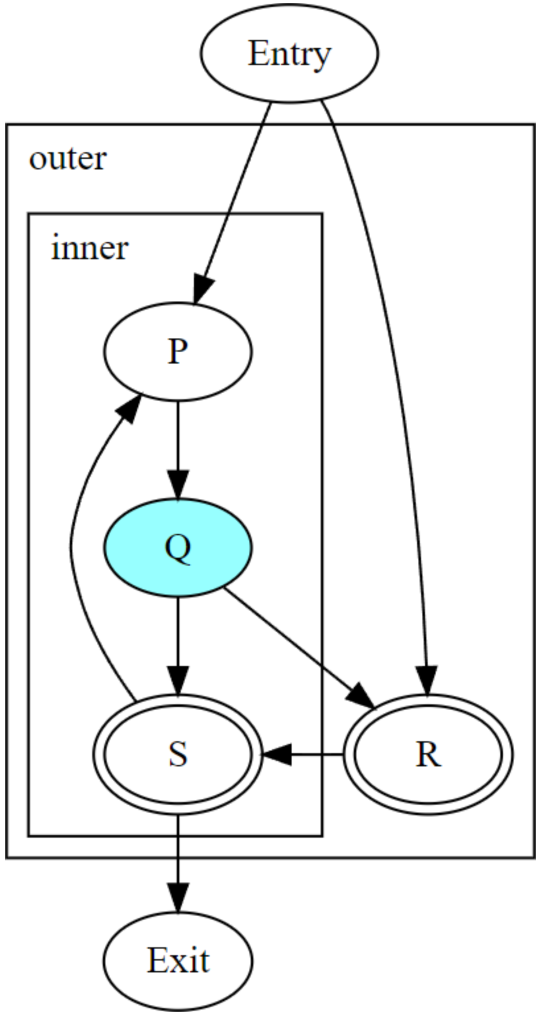
Maximal Convergence: An instance of `converged-with`

- **Expectation 1:** Threads *should* converge as often as possible:
  - At a convergent operation.
  - At the header of a cycle (generalization of a natural loop; can be irreducible).
  - At a post-dominator.
- **Expectation 2:** When threads enter a cycle:
  - Threads may divergently exit the cycle on different iterations.
  - All threads must finish that cycle before reconverging outside.
- Formally captured as **maximal convergence:**
  - Suitable for existing targets.
  - Compatible with the `convergent` attribute.
  - Works with irreducible CFGs.

Maximal Convergence (Informally)

		1	2	3	4	5	6	7	8	9	
Thread1	Entry1	P1	Q1	R1	S1	P3	Q3	S3			Exit1
Thread2	Entry2	P2	Q2	R2	S2	P4	Q4		R3	S4	Exit2

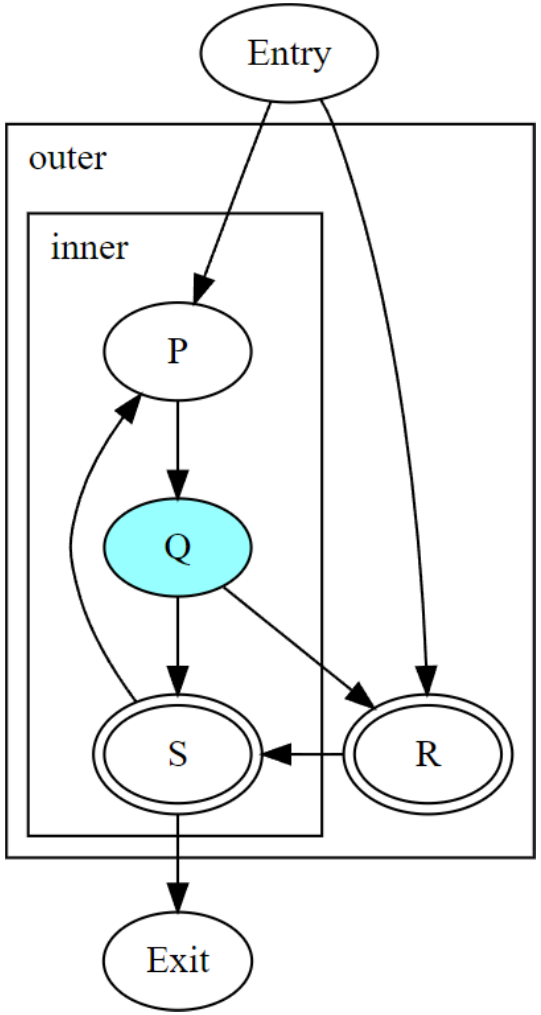
- R and S are cycle headers.
  - Each execution of a header marks a new iteration of the cycle.
- P1 `converged-with` P2 but not with P4 (different iterations).
- S3 not `converged-with` S4 (different iterations).



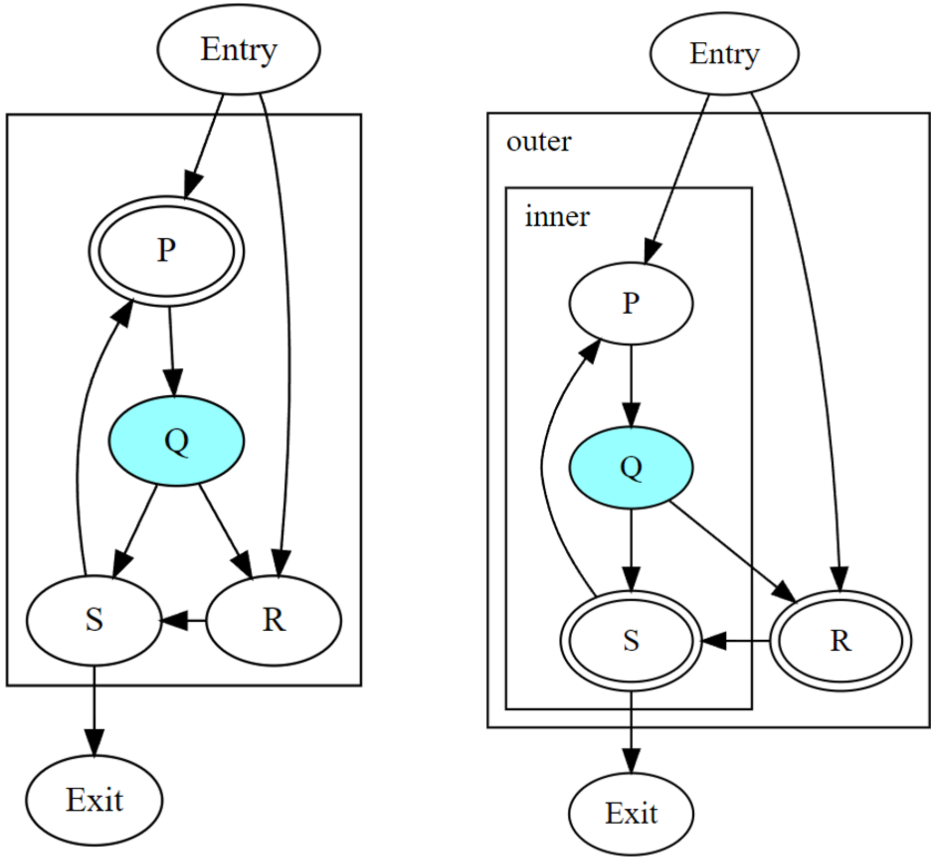
Maximal Convergence (Formally)

		1	2	3	4	5	6	7	8	9	
Thread1	Entry1	P1	Q1	R1	S1	P3	Q3	S3			Exit1
Thread2	Entry2	P2	Q2	R2	S2	P4	Q4		R3	S4	Exit2

- P1 not converged-with P4:
  - Header R2 precedes P4 in the thread but not convergence-before P1.
- S3 not converged-with S4:
  - Header R3 precedes S4 in the thread but not convergence-before S3.
- Dynamic instances X1 and X2 are converged if and only if:
  - for every cycle that contains static instance X, there is no dynamic instance H’ of the header H such that
  - H’ precedes X1 (respectively, X2) in the same thread, and,
  - H’ is not convergence-before X2 (respectively, X1).



Convergence in Irreducible CFGs



- An irreducible CFG can be resolved into cycles in different ways.
  - Each cycle hierarchy produces its own convergence.

Case 1: A single cycle with header P

		1	2	3	4	5	6	7	8	9
Thread1	Entry1	P1	Q1	R1	S1	P3	Q3	R3	S3	…
Thread2	Entry2	P2	Q2		S2	P4	Q4	R2	S4	

Case 2: Nested cycles with headers R and S

		1	2	3	4	5	6	7	8	9
Thread1	Entry1	P1	Q1				R1	S1	P3	…
Thread2	Entry2	P2	Q2	S2	P4	Q4	R2	S4		

M-Converged Static Instances

- A static instance X is **`m-converged`** if and only if
  - Its dynamic instances are **`converged`** **in the same way in every cycle hierarchy.**
- For reducible CFGs:
  - Unique loop hierarchy.
  - All static instances are `m-converged`.
- For irreducible CFGs:
  - Identify certain static instances as `m-converged`.
  - Based on “closed paths” in the CFG, which are independent of cycles.

Uniformity for Irreducible CFGs

- If a static instance is not `m-converged`, outputs are assumed to be `divergent`.
- If a static instance X is `m-converged`, then the outputs are `uniform` if:
  - The semantics of the instruction specifies the output to be `uniform`, OR
  - Each incoming value is uniform, AND
    - If X is a PHI node, then `converged` threads choose the same incoming value.

Implementation

- RFC posted as review on the LLVM Phabricator website:
  - https://reviews.llvm.org/D130746
- The analysis is implemented as a template that can be instantiated for both LLVM IR and Machine IR.
- Current status:
  - Passes existing tests for divergence analysis
  - Passes new tests with irreducible control flow
  - Currently working on Machine IR tests

COPYRIGHT AND DISCLAIMER

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD