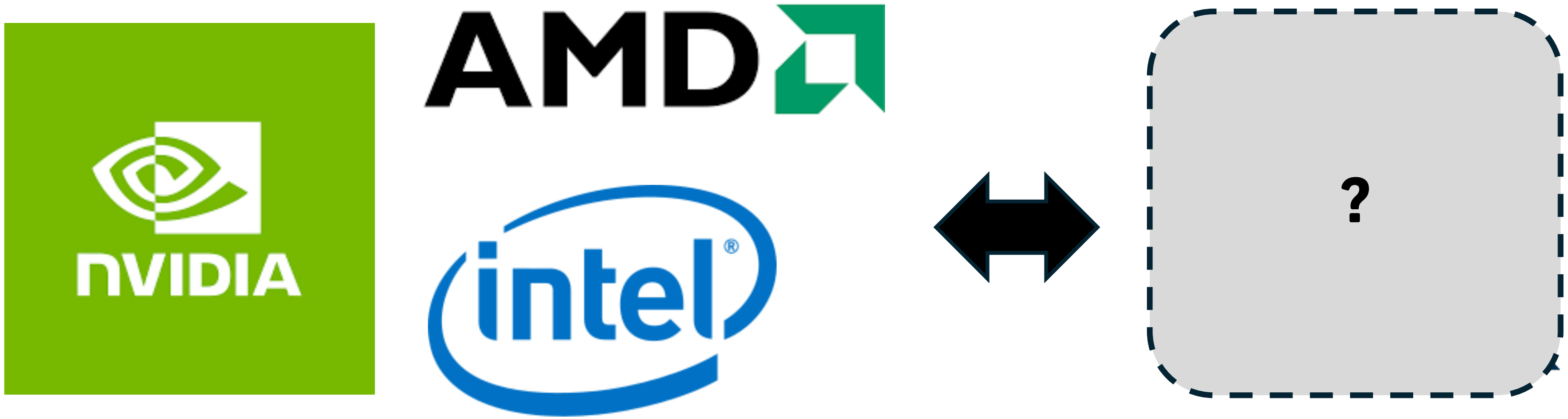
Inside VOLT: Designing an Open-Source GPU Compiler

Shinnung Jeong[1], Chihyo Ahn[1], Huanzhi Pu[1], Jisheng Zhao[1], Hyesoon Kim[1], Blaise Tine[2]
[1]Georgia Institute of Technology, [2]University of California, Los Angeles

GPUs Power Modern Computing - Openness Remains Limited

Core GPU microarchitecture and compiler internals remain largely inaccessible to researchers

What Research Needs Beyond Exposed APIs



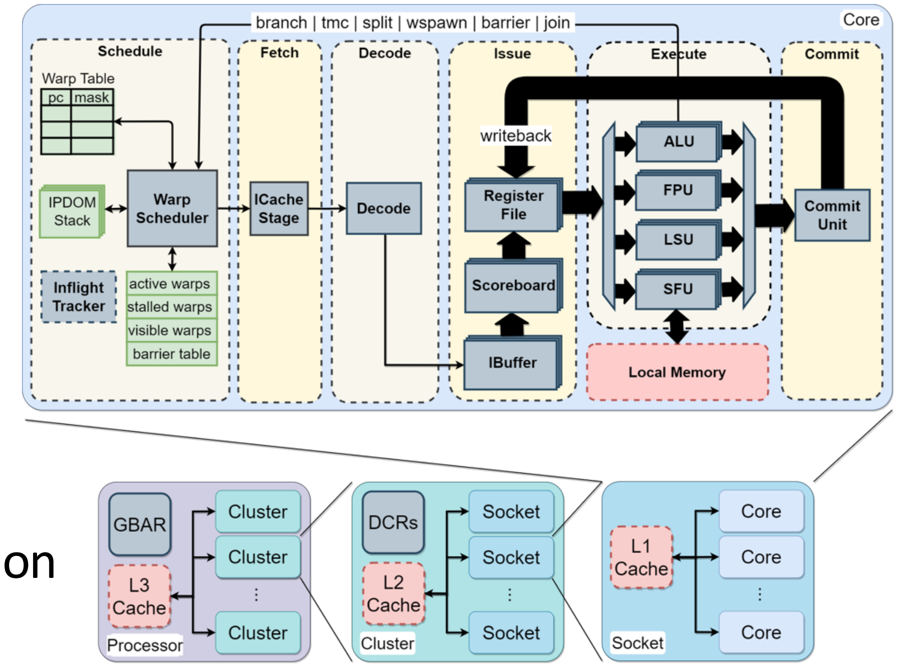
GPU research would benefit from openness beyond exposed APIs

What Research Needs Beyond Exposed APIs

Vortex GPU provides a fully open-source GPU platform

Vortex GPU

- RISC-V based GPGPU
- Highly reconfigurable hierarchical architecture
- SIMT execution model
- Explicit hardware support for SIMT behavior, including control-flow divergence and barriers
- Drivers for host-device communication

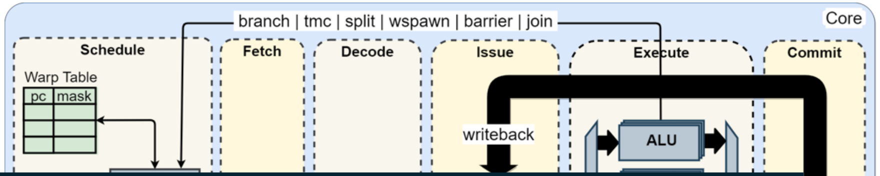


Vortex GPU

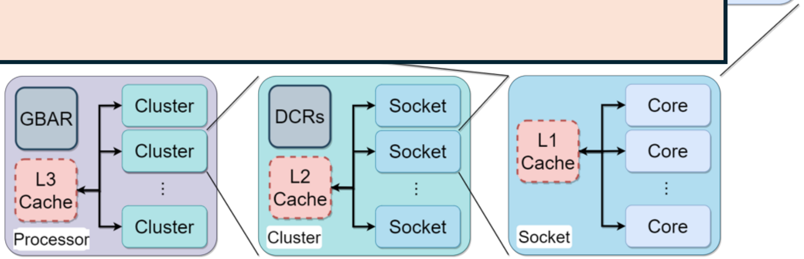
- RISC-V based GPGPU
- Highly reconfigurable hierarchical

divergence and barriers

- Drivers for host-device communication



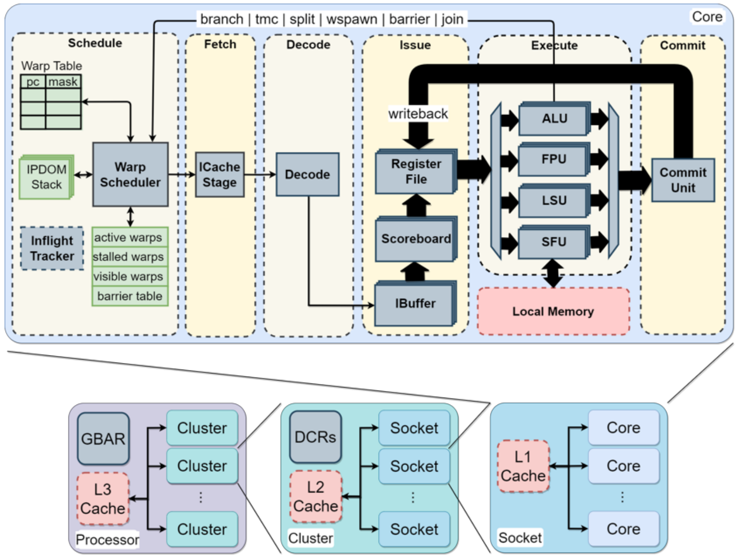
Open GPU hardware is advancing rapidly, while compiler support remains underexplored



Program Execution on Vortex GPU

```
__kernel void foo(){
   tid = get_global_id(0);
   ...
   barrier(...);
}
```

```
__host int main(){
   ...
   cl_mem A_clmem = clCreateBuffer(context, ...);
   clEnqueueNDRangeKernel(command_queue, ...);
   ...
}
```



Program

- Kernel code with control flow & special func
  - Host Code with device communication

Vortex GPU

- Kernel binary using the Vortex GPU ISA
  - Host executable using Vortex driver

Growing GPU Variants on Vortex

```
__kernel void foo(){
   tid = get_global_id(0);
   ...
   barrier(...);
}
```

```
__host int main(){
   ...
   cl_mem A_clmem = clCreateBuffer(context, ...);
   clEnqueueNDRangeKernel(command_queue, ...);
   ...
}
```

Virgo

SkyBox

Tensor core

SparseWeaver

Tapeout

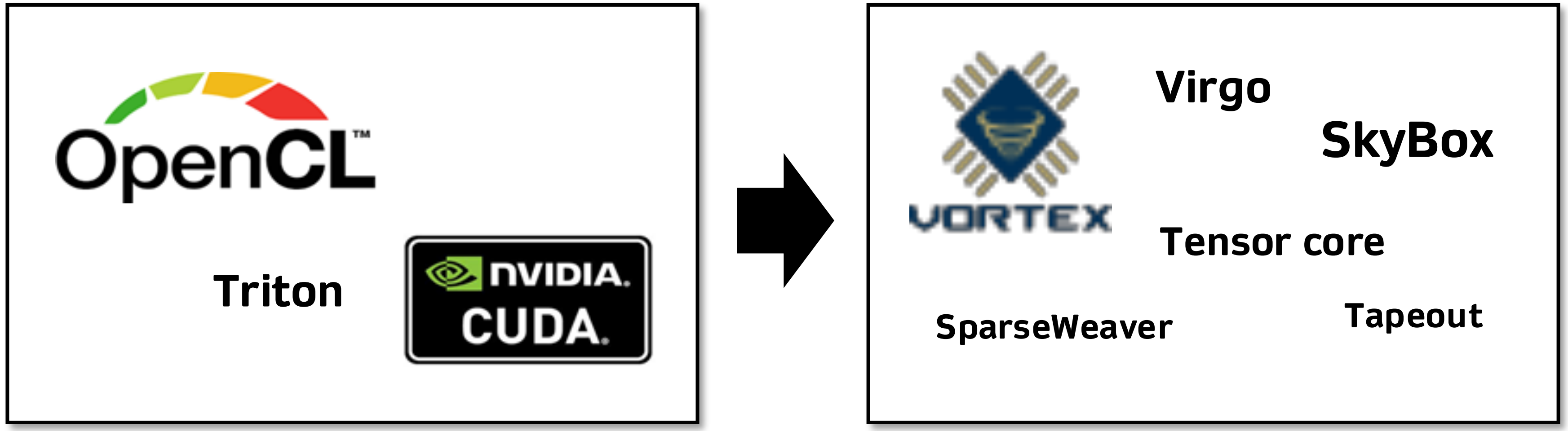
Program

- Kernel code with control flow & special func
  - Host Code with device communication

Vortex GPU Variant

- Kernel binary using the **Extended** Vortex GPU ISA
  - Host executable using **Extended** Vortex driver

Programs from Diverse Frontends



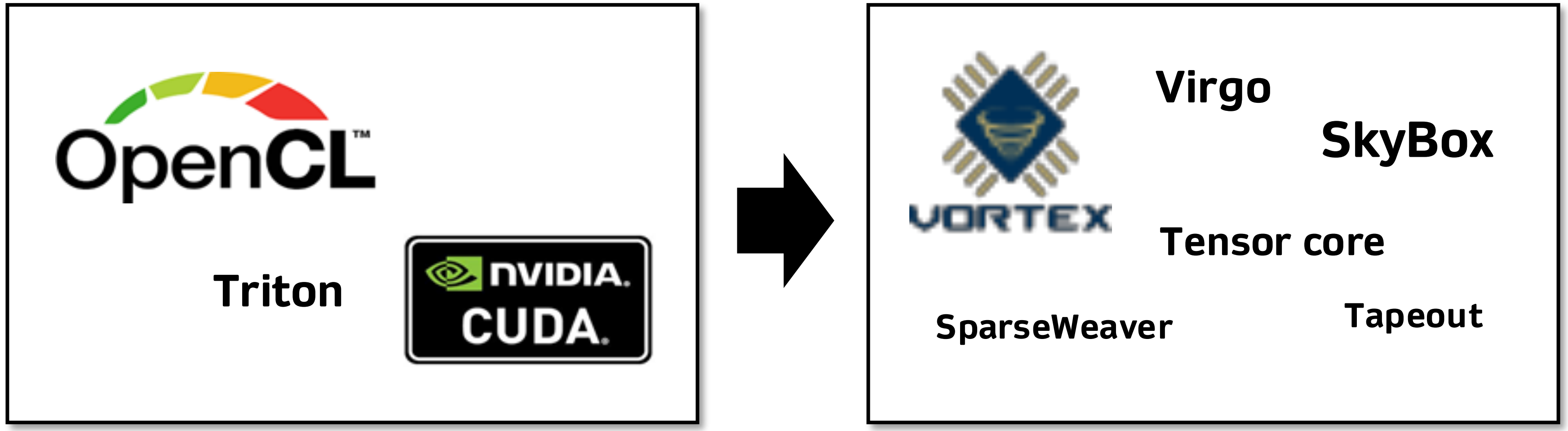
Program with Diverse Frontend

- Kernel code with **Frontend-specific semantics**
- Host Code with **Frontend-specific runtimes**

Vortex GPU Variant

- Kernel binary using the **Extended** Vortex GPU ISA
- Host executable using **Extended** Vortex driver

Managing Growing GPU Variants and Diverse Frontends



Program with Diverse Frontend

Vortex GPU Variant

This Calls for a Well-Designed Compiler Framework

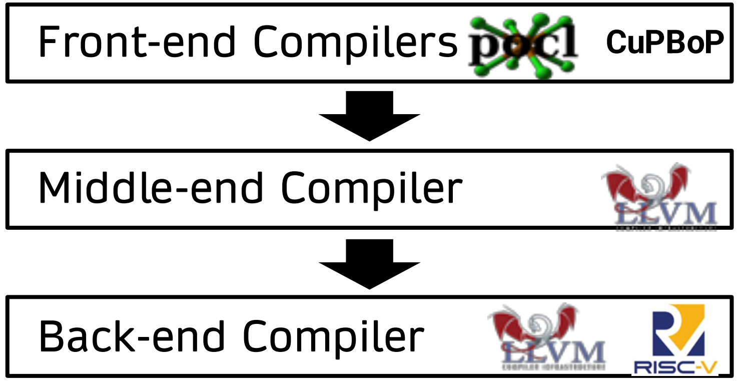
Key Challenges in Designing VOLT

- **Challenge 1: Make SIMT-aware code generation and optimizations portable**
  - How can we preserve SIMT-aware code generation and optimizations while remaining portable across Vortex variants and potentially other open GPUs?

- **Challenge 2: Support multiple frontends with minimal maintenance overhead**
  - How can we design the compiler so that new frontends and GPU variants can be added with minimal refactoring?

Vortex-Optimized Lightweight Toolchain (VOLT)

- **Design Choice 1**
  - Hierarchical compiler design
  - Centralize fundamental SIMT-related analyses and optimizations in the middle-end
- **Design Choice 2**
  - Leverage existing open-source components as much as possible

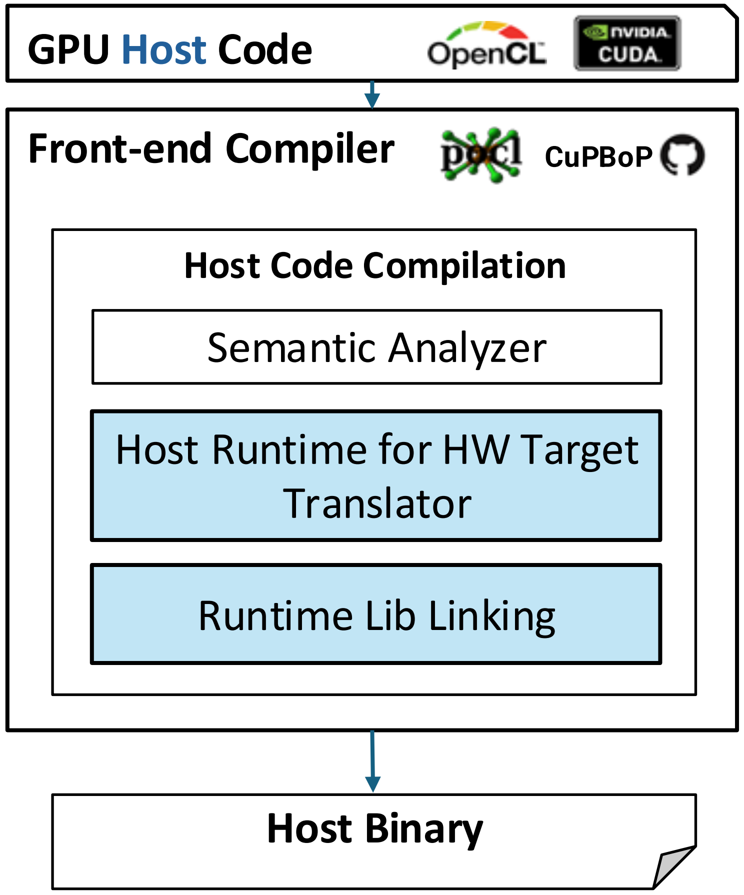


Hierarchical Compiler Design

Front-end Compiler

- Handles Both Host and Kernel Code

- **Host Code**
  - Compiled for the host hardware
  - Translates the frontend host functions using the target runtime library

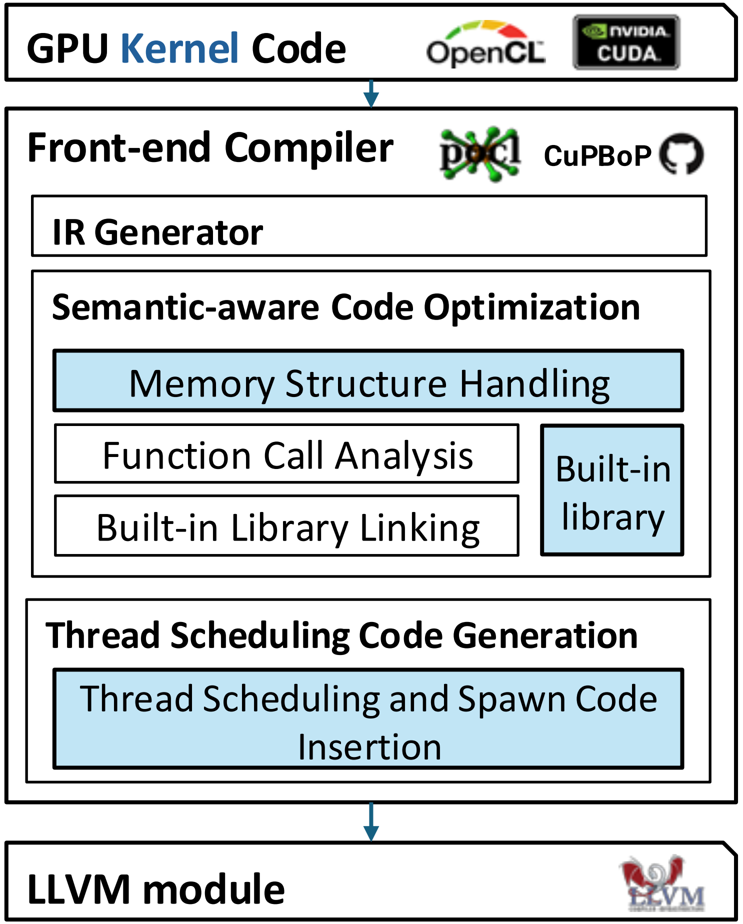


Blue boxes indicate components extended or implemented for the Vortex GPU

Front-end Compiler

- Handles Both Host and Kernel Code

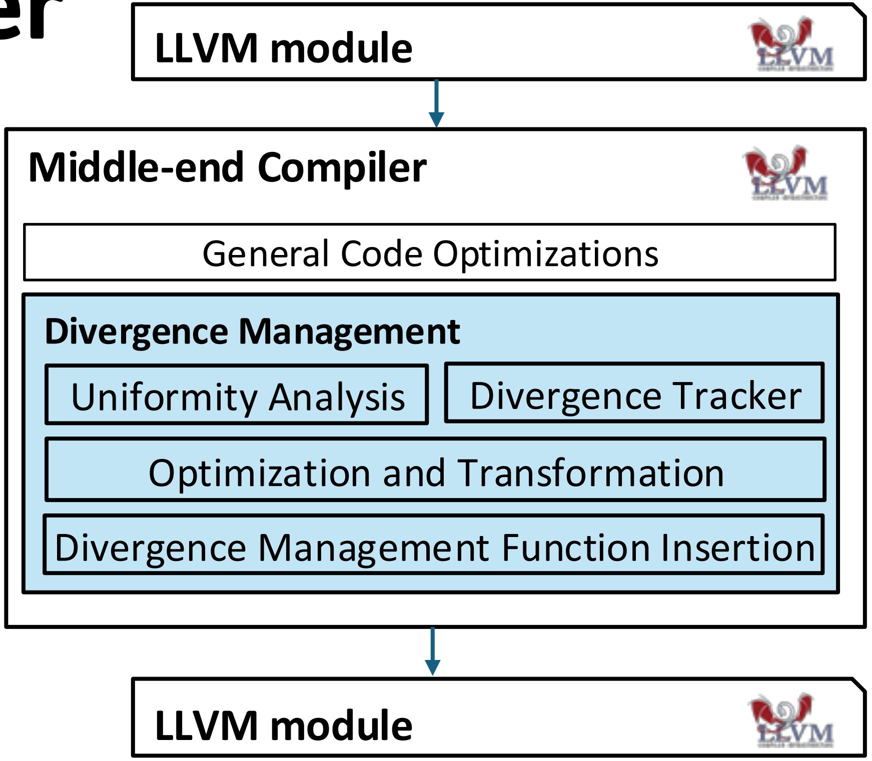
- **Kernel Code**
  - Performs semantics-aware code optimizations
  - Transforms special functions using built-in libraries
  - Inserts thread scheduling and spawning code for device initialization



Blue boxes indicate components extended or implemented for the Vortex GPU

Middle-end Compiler

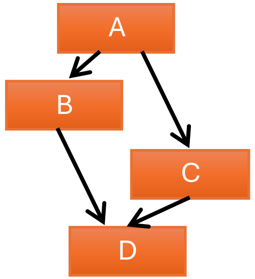
- Target-independent optimizations
- Minimal target-specific logic
- Thread Divergence management treated as a first-class concern



Blue boxes indicate components extended or implemented for the Vortex GPU

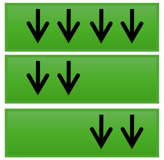
Thread Divergence Management

- **SIMD**
  - Single instruction applied to multiple data elements
  - Uniform control flow
  - Divergent branches are serialized
- **SIMT**
  - Threads execute in lockstep under a shared instruction stream
  - Each thread has its own registers and stack
  - Control-flow divergence is handled via masking and serialization



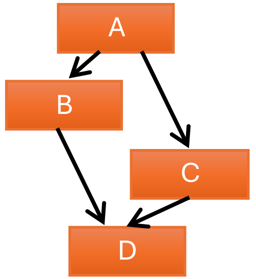
Divergent branch!

```
If (threadid.x<2) {
  work B
} else {
  work C
}
```

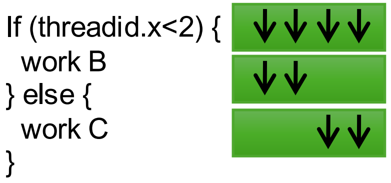


Thread Divergence Management

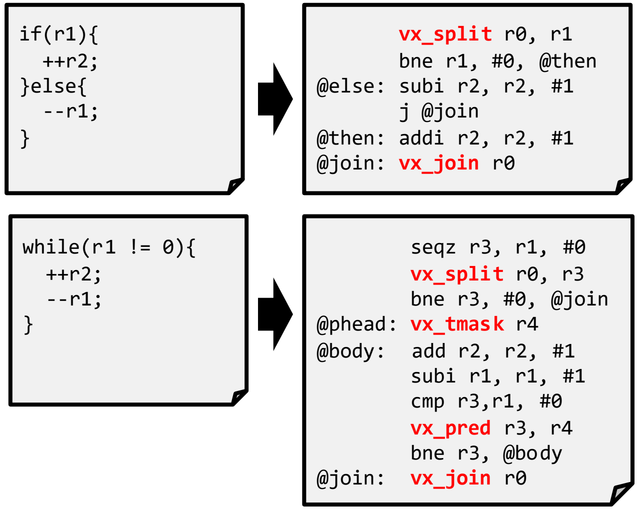
- **Divergence management microarchitecture**
  - IPDOM stack for handling divergence and reconvergence
  - Per-warp thread mask for control-flow management
- **ISA support**
  - Thread mask control instruction
  - Split and join instructions for If
  - Predicate instruction for loop



Divergent branch!



```
If (threadid.x<2) {
  work B
} else {
  work C
}
```



```
if(r1){
  ++r2;
}else{
  --r1;
}
```

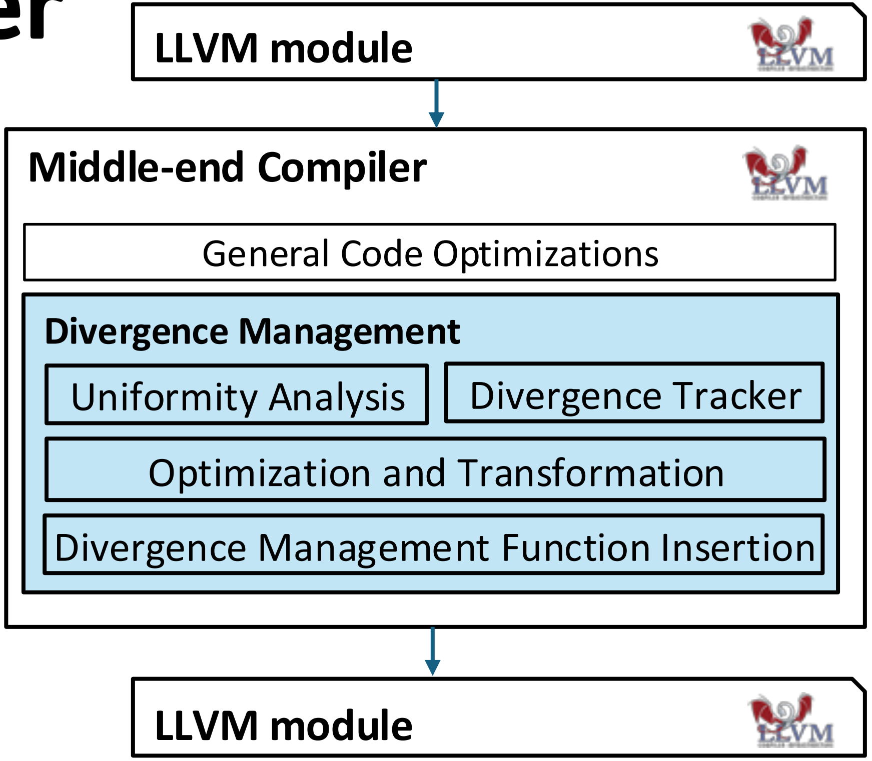
```
        vx_split r0, r1
        bne r1, #0, @then
@else:  subi r2, r2, #1
        j @join
@then:  addi r2, r2, #1
@join:  vx_join r0
```

```
while(r1 != 0){
  ++r2;
  --r1;
}
```

```
         seqz r3, r1, #0
         vx_split r0, r3
         bne r3, #0, @join
@phead:  vx_tmask r4
@body:   add r2, r2, #1
         subi r1, r1, #1
         cmp r3,r1, #0
         vx_pred r3, r4
         bne r3, @body
@join:   vx_join r0
```

Middle-end Compiler

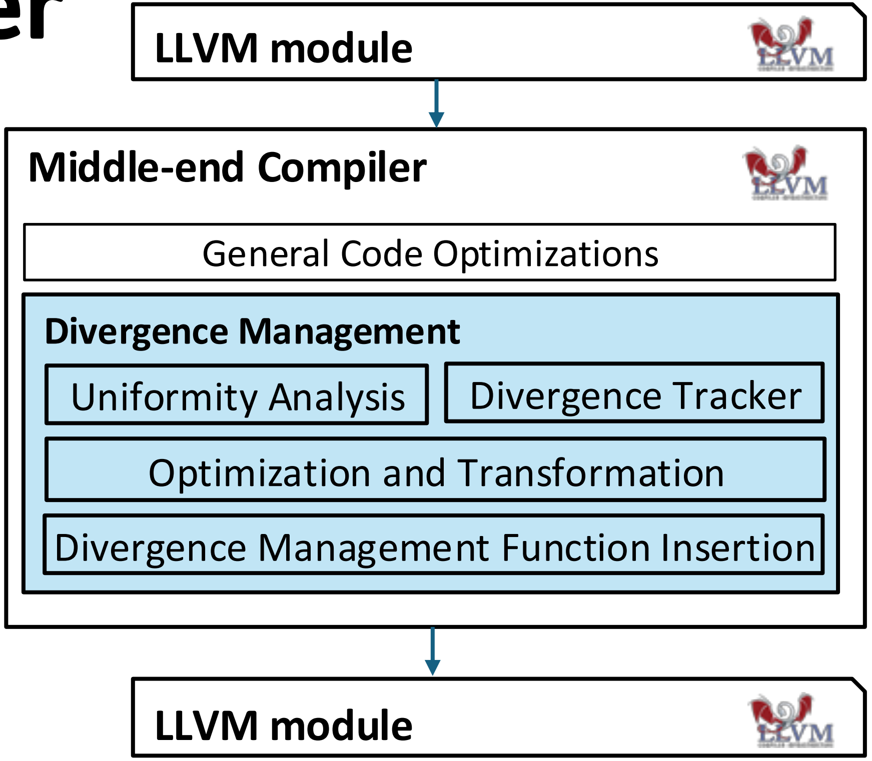
- **Uniformity Analysis**
  - Find divergent instructions and always uniform instructions
  - **Divergence Tracker**
    - Mark Function arg, returns, Atomic as Divergent
    - Mark Control and Status Register info as Uniform
    - Propagate information



Blue boxes indicate components extended or implemented for the Vortex GPU

Middle-end Compiler

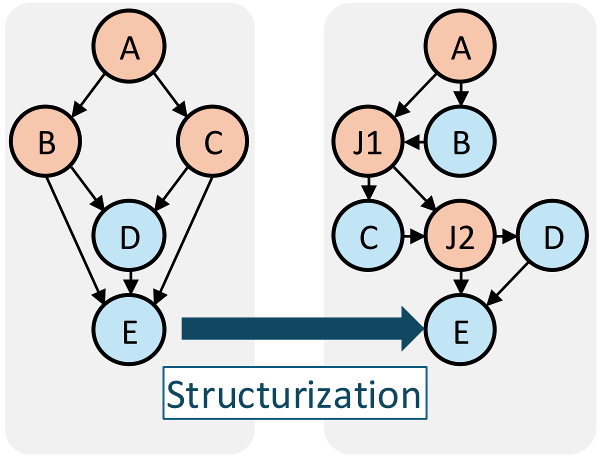
- **Uniformity Analysis**
  - Identify divergent instructions and always-uniform instructions
  - **Annotation analysis**
    - Leverage annotation information from the user or the front-end compiler
  - **Function argument analysis**
    - Determine the uniformity of each function's arguments



Blue boxes indicate components extended or implemented for the Vortex GPU

Middle-end Compiler

- **Divergence Optimization**
  - **Code Simplification**
    - Simplifies control flow
    - Canonicalizes control flow to simplify subsequent divergence handling
  - **Control-Flow Structurization**
    - Transforms the CFG into a structured form to provide stable join points
    - Essential for IPDOM-based divergence and reconvergence handling

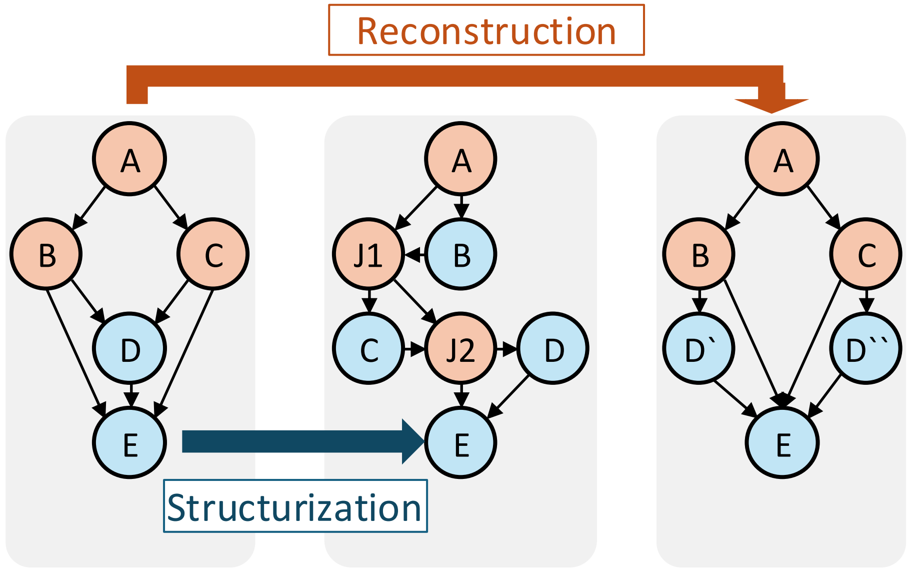


Original CFG

Structurized CFG

Middle-end Compiler

- **Divergence Optimization**
  - **Control-Flow Reconstruction**
    - Selectively duplicates basic blocks to reduce predicate computation
  - **Divergence Operation Lowering**
    - Normalizes divergence operations by rewriting them into equivalent branch-based control flow



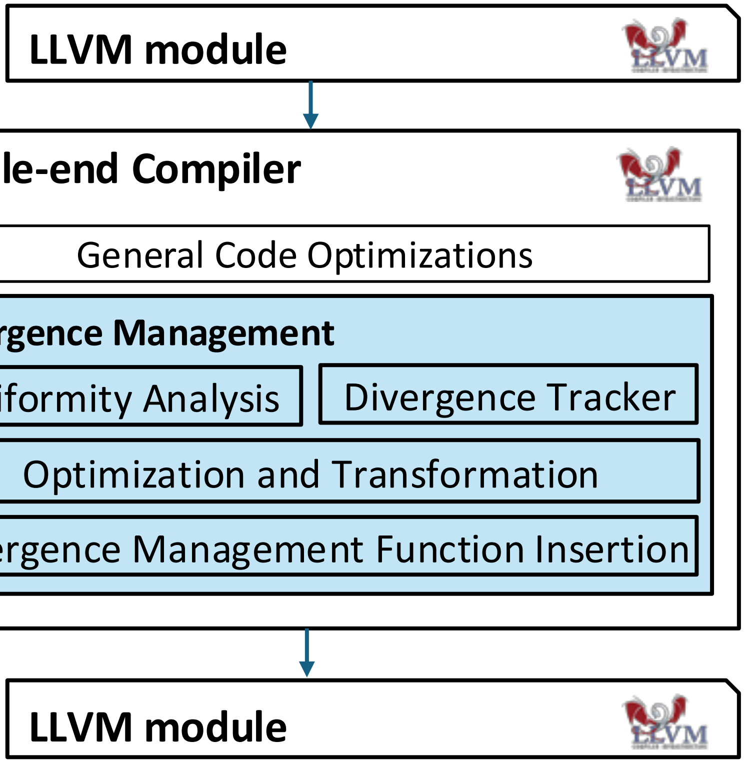
Original CFG

Structurized CFG

Reconstructed CFG

Middle-end Compiler

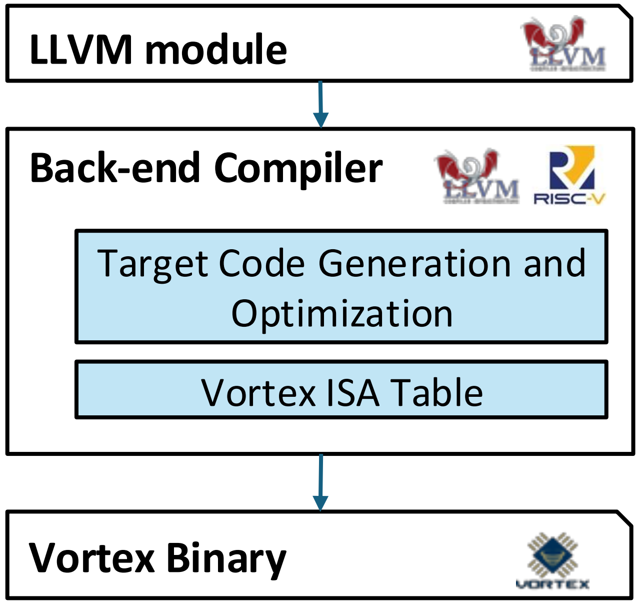
- **Divergence Intrinsic Insertion**
  - **Split and Join Insertion**
    - Handling divergent branch with **vx_split** and **vx_join** instruction
  - **Loop predicate Insertion**
    - Handling divergent loops with **vx_pred**



Blue boxes indicate components extended or implemented for the Vortex GPU

Back-end Compiler

- Handles target-specific optimizations and final code generation
- Built on the RISC-V target compiler
- Extends the ISA table with Vortex-specific ISA extensions
- Generates Vortex target binaries

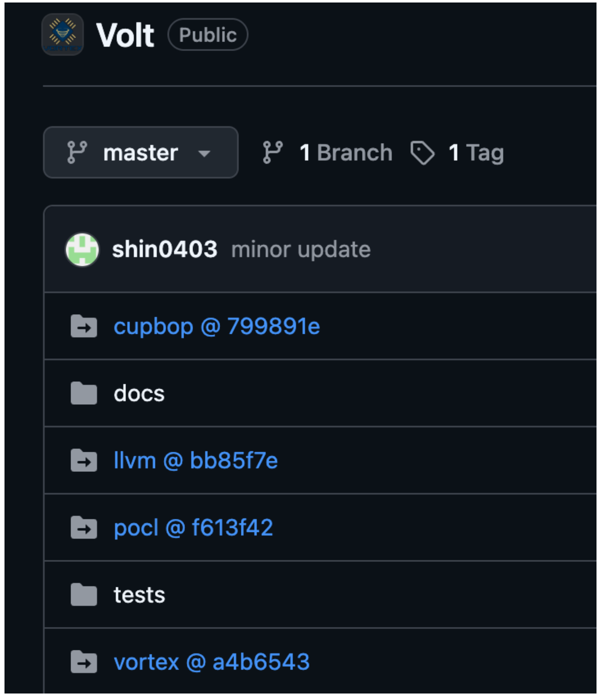


Blue boxes indicate components extended or implemented for the Vortex GPU

VOLT v1.0 Release

- Our tool is available in our Git repository
  - https://github.com/vortexgpgpu/Volt/releases/tag/v1.0

- The first release supports CUDA 12.1 and OpenCL 3.0
- Provides an end-to-end PoCL / CuPBoP / LLVM compilation pipeline targeting the Vortex GPU


Volt Public

master 1 Branch 1 Tag

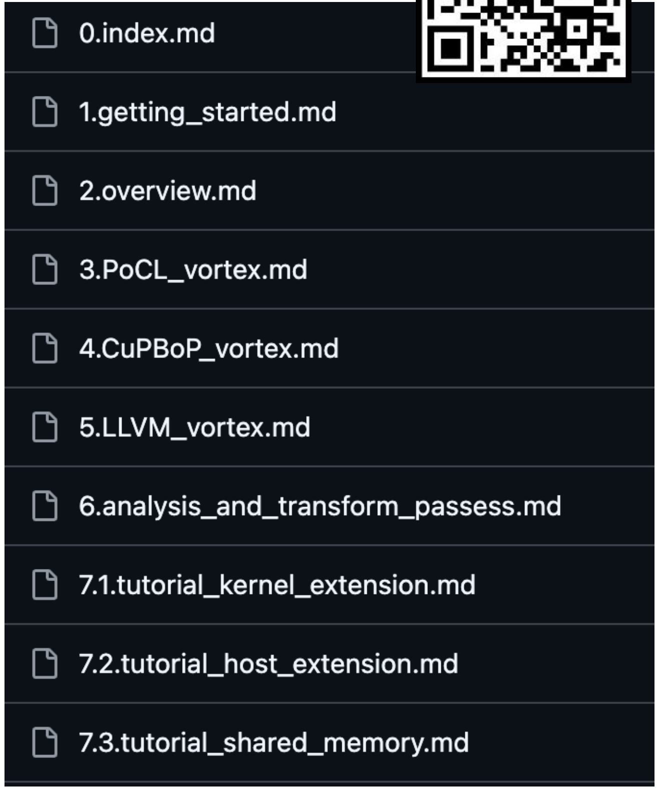
shin0403 minor update

cupbop @ 799891e
docs
llvm @ bb85f7e
pocl @ f613f42
tests
vortex @ a4b6543

Guide for VOLT Extensions

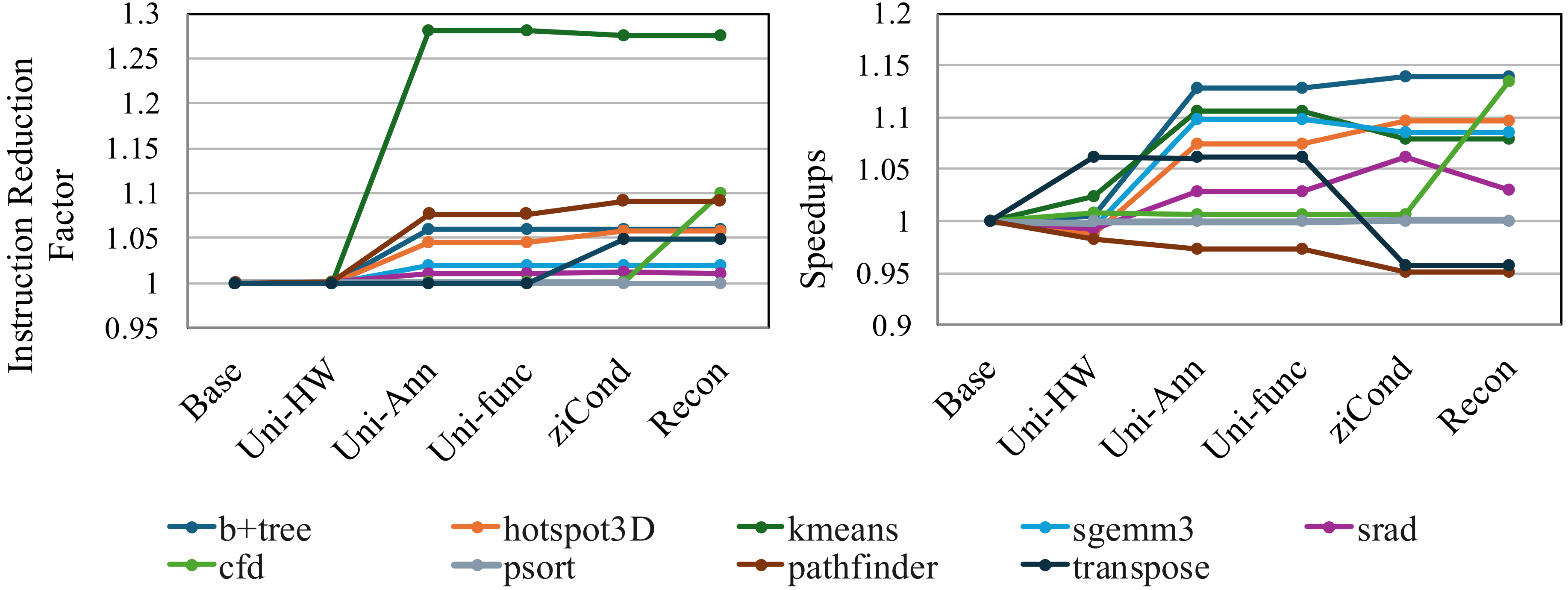
- Tutorial 1: Extending kernel functions
- Tutorial 2: Extending host functions
- Tutorial 3: Extending memory support with the Vortex memory hierarchy

- All tutorials include documentation under /Docs and runnable examples under /Tests.



0.index.md
1.getting_started.md
2.overview.md
3.PoCL_vortex.md
4.CuPBoP_vortex.md
5.LLVM_vortex.md
6.analysis_and_transform_passess.md
7.1.tutorial_kernel_extension.md
7.2.tutorial_host_extension.md
7.3.tutorial_shared_memory.md

Divergence Management Optimization



Research Directions with Volt & Vortex

1. Micro-architectural Compiler Optimization

- Fine-grained instruction & warp scheduling
- Latency hiding via instruction prefetching
- Adaptive workload distribution and occupancy control

2. Architecture-aware Optimization & Analysis

- Exploitation Heterogeneous core
- Explore reconfigurable options

3. HW / SW Co-design Exploration

- Co-design of custom GPU extensions and compiler support
- ISA or micro-architectural features
- Feedback-driven optimization between hardware, compiler, and runtime

Conclusion

- **VOLT is a lightweight, extensible compiler toolchain optimized for Vortex**
- **Hierarchical design, Centralize SIMT-aware optimizations in the middle-end**
  - Enables portability across Vortex variants
- **Thread divergence management is treated as a first-class concern**
  - Uniformity analysis, divergence tracking, and control-flow optimizations
  - Improves performance and reduces instruction count
- **Extensibility and reproducibility**
  - Clear extension points for kernel, host, and memory support
  - Tutorials, documentation, and shared scripts enable easy adoption and extension

Thank you