Implementing SPMD control flow in LLVM using reconverging CFGs

Fabian Wahlster
Technische Universität München – UX3D

Nicolai Hähnle
AMD
Divergence on wide SIMD

Src: D. Lively and H. Gruen. “Wave Programming in D3D12 and Vulkan”

Src: A. Sabne, P. Sakdhnagool, and R. Eigenmann
“Formalizing Structured ControlFlowGraphs.”
Converting thread-level code to wave-level ISA

```c
float fn0(float a, float b) {
    if (a > b)
        return ((a - b) * a);
    else
        return ((b - a) * b);
}
```

// Registers r0 contains "a", r1 contains "b"
// Value is returned in r2

```
v_cmp_gt_f32 r0, r1  // a > b, establish VCC
s_mov_b64    s0, exec // Save current exec mask
s_and_b64    exec, vcc, exec  // Do "if"

s_cbranch_vccz label0  // Branch if all lanes fail
v_sub_f32    r2, r0, r1 // result = a - b
v_mul_f32    r2, r2, r0 // result = result * a

label0:

s_andn2_b64  exec, s0, exec  // Do "else" (s0 & !exec)

s_cbranch_execz label1  // Branch if all lanes fail
v_sub_f32    r2, r1, r0 // result = b - a
v_mul_f32    r2, r2, r1 // result = result * b

label1:

s_mov_b64    exec, s0  // Restore exec mask
```

M. Mantor and M. Houston: AMD Graphic Core Next Architecture, Fusion 11 Summit presentation
Structurization in LLVM

StructurizeCFG pass

Unnecessary flow blocks
Reconverging CFGs

Definition:
• Every non-uniform terminator B (conditional branch) has exactly two successors
• One of which post-dominates B
Lowering Reconverging CFGs

For each conditional non-uniform node $N$:

- Virtual register $m$ holds re-join mask for basic block $N$
- Subtract $m$ from the exec register to direct control flow to secondary successor
- Add $m$ the exec register at the beginning of the primary successor to re-join divergent threads
- $m$ must be correctly initialized to avoid unrelated data being merged into the execution mask
Transforming to reconverging control flow

Approach:
- Maintain open tree OT structure containing unprocessed open edges to reroute control flow towards the exit node by inserting new flow blocks

Ordering:
- Compute basic block ordering in which to process input CFG
- Ordering is based on traversal of the input CFG
- Any ordering is viable as long as the exit node comes last
- Quality of reconverging CFG depends on the input ordering
Open Tree Structure

Processing nodes:
- Nodes of the OT have sets of open *Incoming* and *Outgoing* edges that need to be processed
- An outgoing edge \((A, B)\) is closed if \(A\) has already been visited when \(B\) is being processed
- A node can be closed if both sets are emptied by processing
- Closed nodes are removed from the OT and their child nodes moved to its parent
- Divergent nodes are called *armed* if one of the outgoing edges has already been closed
Transforming to reconverging control flow

Input CFG

A → C → B → D

Initialize OT

ROOT

Added A

ROOT

Processing C...

ROOT

A

C

B

B

C

D
Transforming to reconverging control flow

Input CFG

Processing C…

Adding FLOW

Output CFG
Transforming to reconverging control flow

Reconverging CFG

A:
%cc_A = icmp eq i32 %in_A, 0
br i1 %cc_A, label %FLOW0, label %C

B:
br label %D

FLOW0:
%0 = phi i1 [ true, %A ], [ false, %C ]
br i1 %0, label %B, label %D

C:
%cc_C = icmp eq i32 %in_C, 0
br i1 %cc_C, label %A, label %FLOW0

D:
ret void
Input Ordering Exit Condition

Input CFG

Depth First:

Input: A → B → D → F

Depth First: A → B → D → F → E → C

Exit Condition: A → B → D → F

Exit Condition: A → B → D → F → E → C
Input Ordering Comparison

Depth First:

Breadth First:

RPOT:
Reconverging Control-Flow Graphs

Contributions:

• New SPMD vectorization approach
• Simple and concise definition of Reconvergence for CFGs (weaker than structuredness)
• Proof-of-Concept lowering algorithm and CFG transformation

Properties:

• Support for unstructured and irreducible input CFGs
• Preserves uniform control flow
• Retains CFGs that are already reconverging
• Insert fewer new basic blocks than structurization (StructurizeCFG)