Implementing SPMD control flow in LLVM using reconverging CFGs

Vectorizing divergent control flow for SIMD applications

Fabian Wahlster
Technical University Munich Department of Informatics
fwahlster@outlook.com — 1 (000) 111 1111

Abstract
Compiling programs for an SPMD execution model, e.g., for GPUs or for whole program vectorization on CPUs, requires a transform from the thread-level input program into a vectorized wave-level program in which the values of the original threads are stored in corresponding lanes of vectors. The main challenge of this transform is handling divergent control flow, where threads take different paths through the original CFG. A common approach, which is currently taken by the AMDGPU backend in LLVM, is to first structure the program as a simplification for subsequent steps. However, structurization is overly conservative. It can be avoided when control flow is uniform, i.e., not divergent. Even when control flow is divergent, structurization is often unnecessary. Moreover, LLVM’s StructurizeCFG pass relies on region analysis, which limits the extent to which it can be evolved. We propose a new approach to SPMD vectorization based on saying that a CFG is reconverging if for every divergent branch, one of the successors is a post-dominator. This property is weaker than structurization, and we show that it can be achieved while preserving uniform branches and inserting fewer basic blocks than structurization requires. It is also suitable for code generation, because it guarantees that threads which “leave” a wave at divergent branches will be able to rejoin it later.

Reconverging control flow graphs
We argue that the structurization used in LLVM’s StructurizeCFG region pass is too intrusive with respect to the input control flow. The weaker notion of reconvergence is sufficient for re-joining diverging threads required to generate wave-level code, while also handling uniform control flow properly.

Definition 1. A control flow graph is reconverging if every non-uniform condition node (terminator $T$) has exactly two successors, one of which post-dominates it (primary successor).

Figure 1: Reconverging CFGs are suitable for lowering to wave-level control flow by inserting instructions for execution mask manipulation.

The simple definition of reconverging CFG introduced in this work shows that it’s powerful enough to effectively lower thread-level to wave-level control flow, solving the common problem of divergent control flow graphs in code generation for SPMD and SIMT applications. Listing 1 shows the results of the lowering algorithm based on a reconverging input CFG.

Listing 1: Pseudo-GCN ISA pseudocode for control flow of Figure 2a

1. $v_{emp.
2.各级\( s[i][j] \) ... // Initialize re-join mask in $B$
3. $v_{emp.
4. $v_{emp.
5. $v_{emp.
6. $s[i][j]$ // code for $B$
7. $v_{emp.
8. $v_{emp.
9. $v_{emp.
10. $v_{emp.
11. $v_{emp.
12. $v_{emp.
13. $v_{emp.
14. $v_{emp.

Main Contributions
- Concise definition of a reconverging CFG
- Lowering algorithm exploiting the properties of such a CFG
- Algorithm transforming arbitrary and irreducible input CFGs to contain reconvergence points while also preserving uniform control flow
- Evaluation of basic block ordering methods used as input of the CFG transformation

Algorithm
The algorithm for vectorizing divergent control flow using the properties of reconverging CFGs consists of the following steps:

1. Make sure a unique sink exists by adding a common virtual exit to be able to merge control flow from non-uniform branches that lead to different exit nodes.
2. Determine an ordering of the basic blocks based on the topology of the CFG. The correctness of the following transformation is not affected by the ordering created in this step, but the quality of the resulting CFG depends on it.
3. Transform the control flow to ensure existence of reconvergence points by adding flow blocks and rerouting edges accordingly.
4. Inject predication operations to lower the reconverging CFG to wave-level. Optionally keep scalar branch instructions to skip blocks that would be executed with an empty execution mask.

OpenTree structure
The transformation of input CFGs is executed on a bookkeeping structure called OpenTree, short OT. Open (dotted) edges in the OT are initialized with edges from the input CFG but may change (rerouted) over the course of the algorithm. Solid light gray edges are closed. Visited nodes have a black frame, unvisited nodes light gray. Solid black edges symbolize OT parent-child edges. Divergent nodes are called armed (red) if one of the outgoing edges has already been closed.

Input orderings
The input ordering of basic blocks affects the quality of generated control flow as it changes when and how the critical edges are detected and rerouted while processing the nodes of the OpenTree.

Figure 2: CFG of Figure 2a incurs Definition 1. CFG of Figure 2b is suitable for lowering to wave-level.

Figure 3: The instructions need to be inserted at flow-blocks to maintain original control flow.