VLoop in VeGen

1 Loop Related Files

There are three groups of files that are directly related to loops in VeGen: Vloop, LoopUnrolling, and UnrollFactor. Each group has a header file and a cpp file.

2 VLoop

2.1 Members

VLoop is built upon the original LLVM loop and contains a pointer to the original loop. It has references to two control conditions: one decides whether to execute the loop, and the other is the back edge condition, which is essentially the loop condition in C but contains a chain of conditions from the function entry.

VLoop also contains a reference to the VectorPackContext. We can see a design pattern here: an object often has references to its context and to other supportive data structures that belong to the context. Here, the context includes the GlobalDependenceAnalysis and the VLoopInfo (which is a friend class of VLoop). VLoop has bit vectors to store the instructions that depend on the loop and the instructions that are contained in the loop, as well as a reference to the subloops.

Some condition-related things are here. They are maps from phi nodes to mu nodes, from phi nodes to one-hot phis, and from phi nodes to lists of control conditions (Gated phis). An instruction has a guard value, which is the value that instructions outside of this loop should use. Also, there are conditions for each instruction.

2.2 Methods

2.2.1 haveIdenticalTripCounts

First, try to use the LLVM ScalarEvolution class to obtain the back edge taken count of loop 1 and loop 2. If both are computable and equal, the loop trip counts are identical.

If not, check the exit block of the two loops. Only in this case do we need to consider further: both have an exit block, and the terminators of the exit blocks are identical.

Rely on the ScalarEvolution framework to recognize the loop counter, which is an affine expression of the trip. If the affine expression is equivalent, then the two loops have identical trip counts.

2.2.2 isSafeToFuse

To fuse, the loops should be control-equivalent, in the same loop level, independent (two loops are independent if neither has any common instructions with the other's dependent instructions) and having the same trip counts. Moreover, their parents should be safe to fuse.

2.2.3 isSafeToCoIterate

Loops that execute under different control predicates or have different trip counts cannot be fused, but they can become co-iterate if they are independent of each other. The difference in control can be resolved in the loop body by using any control statement, and the difference in trip count is also a form of control that depends on the loop variable. Therefore, the only prerequisite left is the independence of the loop body.

2.2.4 fuse

Fuse is a method of VLoopInfo. First of all, we need to fuse loops in a top-down manner, that is, merge two loops only when their parents are merged. For two loops VL2 and VL1, we try to fuse VL2 into VL1.

It is worth noticing that in VeGen, there is no clear separation between transform pass and analyze pass. I think this causes many analysis results to go into data structures that hold IR, making context, analysis, and LLVM IR jam together.

For example, VLoop defined by VeGen contains not only a reference to LLVM loop and instructions, but also holds dependent analysis results as bit vectors for instructions in the loop and instructions the loop depends on. When fusing two loops, these analyses are updated with fusing. This again makes me think that LLVM project has a clear design.

After fusing, VL2 is added to the deleted loops of VLoopInfo, and erased from the common parent's subloops.

2.2.5 coiterate

VLoopInfo uses an EquivalenceClasses (union-find) ADT to store co-iterated loops.

Coiteration operates on the leaders of the EquivalenceClasses.

3 LoopUnrolling

Loop unrolling replicates a value in multiple iterations of a loop body. VeGen defines UnrolledValue as a data structure to track the value across iterations. It also implements its own version of llvm::UnrollLoop that maintains the unrolled values.

3.1 needToInsertPhisForLCSSA

LCSSA stands for Loop Closed Static Single Assignment. LCSSA ensures that every value defined inside a loop and used outside the loop is assigned to a PHI node at each exit block of the loop. This makes loop analysis and transformations simpler and more effective. If we see a loop as a special function (inlined), then LCSSA is putting the return values of the function loop into the exit block, so they can be read from there.

In VeGen, needToInsertPhisForLCSSA checks all the instructions outside the loop. If any outside loop instruction has an operand which is defined in a loop containing the loop we are checking, then this loop needs to insert phis for LCSSA.

3.2 simplifyLoopAfterUnroll2

3.3 UnrollLoopWithVMap

3.3.1 LoopUnrollResult Unmodified

4 UnrollFactor