Solver in VeGen

The role of the solver in the VeGen framework was initially unclear to me. I discovered that the optimizeBottomUp function, which is invoked in the GSLP driver, is the core of the solver. The solver aims to create a good VectorPackSet for later code generation, using planning and heuristics. The construction of a VectorPackSet is simple, but the codegen method is complex. The only interaction between the solver and the VectorPackSet is the tryAdd method, which checks the compatibility of the VectorPack to be added and updates the PackedValues, AllPacks, and ValueToPackMap.

The main logic of the solver is implemented in another optimizeBottomUp function, which is overloaded with Packer and SeedOperands as parameters and returns a vector of VectorPacks to form a VectorPackSet. The solver is not a class, but a function named optimizeBottomUp in the solver file.

1 optimizeBottomUp

The function optimizeBottomUp is a solver that tries to find the best vectorization plan for a given set of seed operands. It takes four parameters:

- Packs: a reference to a vector of VectorPack pointers, which are data structures that represent a group of scalar instructions that can be packed into a single vector instruction. The function will append the optimal vector packs to this vector.
- Pkr: a pointer to a Packer object, which is a class that handles the packing of vector instructions. It also provides access to the LLVM context and the target machine information.
- SeedOperands: an array of OperandPack pointers, which are data structures that represent a group of scalar operands that can be packed into a single vector operand. These are the starting points for the vectorization process.
- BlocksToIgnore: blocks that should be skipped.

The function performs the following steps:

- It creates a CandidatePackSet object, which is a data structure that stores a set of candidate vector packs and a mapping from scalar instructions to vector packs that contain them.
- It resizes the Inst2Packs vector, which is a member of the CandidatePackSet object, to match the number of values in the VectorPackContext. This vector stores a list of vector packs for each scalar instruction.
- It iterates over the candidate vector packs and updates the Inst2Packs vector accordingly. For each vector pack, it gets the bitset of the elements that are packed, and for each set bit, it adds the vector pack to the corresponding list in the Inst2Packs vector.
- It creates a Plan object, which represents a vectorization plan. It takes the packer as a parameter and initializes the plan with the scalar cost, which is the total cost of executing the scalar instructions without vectorization.
- It calls the improvePlan function, which is defined in the same file as optimizeBottomUp, to find the optimal vectorization plan for the given seed operands and candidate vector packs. This function uses a greedy algorithm that tries to reduce the cost of the plan by adding or removing vector packs, while respecting the dependencies and constraints of the vectorization process.
- It insert the vector packs from the optimal plan to the Packs vector, which is the output parameter of the function.

– It calls the findDepCycle function, which is defined in the same file as optimizeBottomUp, to check if the vectorization plan introduces any dependence cycles that would prevent the correct execution of the program. If so, it prints an error message, clears the Packs vector, and returns the scalar cost as the result of the function.

– Otherwise, it returns the cost of the optimal vectorization plan as the result of the function.

2 Plan

A VPlan is a data structure used by VeGen, a vectorization framework for LLVM. A VPlan contains a reference to a Packer object, a floating-point number Cost, and a set of VectorPack objects. As the name suggests, a VectorPackSet is also a collection of VectorPack objects, but a VPlan is more concerned with the cost and the search algorithm.

3 improvePlan

The function name is somehow misleading, since originally there is no plan at all. It is improvePlan that plans from the beginning.

3.1 Seeds

The improvePlan function takes a vector of VectorPack objects as Seeds.

3.1.1 Store Packs

- It iterates over the instructions in the function obtained from the input Packer, and ignores those that are not scalar store instructions.

- Then, it calls getSeedMemPacks, which uses AccessDAG to pack consecutive scalar stores into a VectorPack. When generating VectorPacks, the function sets the vector length (VL) to 2, 4, or 8, and adds the generated packs to the Seeds.

- Finally, it sorts the VectorPacks in Seeds according to the id in LayoutInfo and the descending order of their element counts, which prioritizes smaller ids (i.e., the leaders in access) and larger element counts.

3.1.2 Loop-Reduction Packs

A loop reduction variable is updated in every loop iteration, so it must be compiled into a φ instruction in the SSA IR. Starting from the φ instructions, the improvePlan function recursively builds a reduction chain, producing a ReductionInfo object. The reduction variable can be either integer or floating-point, and it cannot have more than one use. Moreover, the reduced values that form the reduction variable cannot have more than one in-loop use (otherwise, after vectorizing, we have to extract the value from the vector for that use). The function also takes special care of the operands of the call instructions, trying to find the reduction opportunities of these operands.

3.1.3 Heuristic

The Heuristic module in VeGen is responsible for finding a suitable solution for a given operand pack. The interface of Heuristic has only one important function, namely solve, which takes a constant reference to an OperandPack and returns a Solution object. A Solution object contains an array of VectorPacks that represent the vectorized code (similar to a VPlan).

The Heuristic module is based on three simple rules, which correspond to three heuristic ways of building vector packs. They are: (1) insert all operands into a vector pack, (2) broadcast an operand if all elements in a vector pack are equal, and (3) extract an operand from an existing vector pack.

An OperandPack must be a vector, not a set, because one Value could be used multiple times. To enable more flexibility in using the extracted values, the Heuristic module also allows two shuffle strategies, transpose and de-interleave, to rearrange the elements in a vector pack.

In the Solver module, the Heuristic module has two uses. One is inside the improvePlan function, which is passed to the runBottomUpFromOperand function. The other is inside the tryPackBackEdgeConditions function, which is used to handle loop-carried dependencies.