# Compiler Driver in VeGen

## 1 Pass Plugin

VeGen is implemented as an LLVM pass plug-in, which can be executed using both llvm-opt and clang.

```
clang -Os xxx.c -fpass-plugin=vegen.so -Xclang -load -Xclang vegen.so
```

```
opt -load vegen.so -load-pass-plugin vegen.so -passes=slp -o /dev/null xxx.ll
```

## 2 Intrinsic Wrapper

Inserting backend-specific instructions into a function via copying them directly from a compiled LLVM IR module represents a more straightforward approach compared to other methods.

In particular, VeGen determines whether to insert an instruction according to the semantics of intrinsic functions. Consequently, VeGen first compiles these intrinsics before copying them over to the vectorizing function.

## 3 Header Files

The compiler driver of VeGen is called GSLP. GSLP directly includes the following header files:

- Control Dependence
- Block Builder
- IR Vec
- Inst Sema
- Packer
- Solver
- Unroll Factor
- Vector Pack Set
- Scalarizer

## 4 Command Line Options

- Wrappers Dir: specify the directory of the wrapper functions

- Vectorize Only: only perform vectorization without packing

- Filter: select the loops to be vectorized

- Disable Unrolling: disable loop unrolling

- Disable Cleanup: disable cleanup passes after GSLP

- Disable Reduction Balancing: disable balancing the reduction tree

# 5 GSLP

GSLP is a function pass. It contains a reference to the bitcode of intrinsic wrapper functions and a LLVM target triple storing the architecture type.

GSLP depends on many LLVM analyses, such as Scalar Evolution, Lazy Value Info, Alias Analysis, Dependence Analysis, Dominator Tree, Loop Info, Post Dominator Tree, Target Transform Info, and Block Frequency Info.

## 5.1 Registration

GSLP runs after the following passes: Scalarization, GVN Hoist, Unify Function Exit Nodes, Loop Simplify, Loop Rotate, and LCSSA.

If cleanup passes are not disabled, the following passes will run after GSLP: CFG Simplification, Jump Threading, Instruction Combining, GVN, and Aggressive DCE.

## 5.2 Initialization

When initializing, GSLP gets the architecture type from an input module, then parses the IR file of wrapper functions accordingly.

## 5.3 Run on Function

### 5.3.1 Skip Unsupported Functions

For now, let me ignore the balancing of the reduction tree.

The first thing GSLP does when it runs on a function is to check whether it contains irreducible control flow. If it does, GSLP skips that function. Moreover, every basic block must terminate with a return instruction or a branch instruction, which excludes things like switches and invokes. GSLP also skips infinite loops.

### 5.3.2 Fill up Supported Intrinsics

### 5.3.3 Loop Unrolling

### 5.3.4 BottomUp Packing

### 5.3.5 Code Generation