VeGen: A Vectorizer Generator for SIMD and Beyond

1 Introduction

2 Motivation

3 Lane Level Parallelism

Lane LevelParallelism (LLP) is a relaxation of superword level parallelism (SLP), which models short-vector parallelism (in which an instruction executes multiple scalar operations in parallel) with the following restrictions:

- The operations execute in lock-step.
- The inputs and outputs of the operations reside in packed storage (usually implemented as vector registers). We refer to an element of such packed storage as a lane.

The properties of LLP depend on the semantics of individual instructions. Different instructions can use different combinations of operations or apply different cross-lae communication patterns.

3.1 Non-isomorphism

LLP allows different operations to execute in parallel, whereas SLP applies only one operation across all vector lanes.

3.2 Cross-lane communication

LLP allows an operation executing on one lane to access values from another input lane (as long as the lane is selected statically).

4 Workflow

4.1

4.2

4.3 Pattern Matching

- The result of pattern matching is a match, an IR instruction DAG with possibly multiple live-ins and a single live-out. VeGen represents each match as a tuple consisting of its live-ins, live-out, and operation.
- VeGen records the matched patterns in a match table, $\langle \text{live}_{\text{out}}(m), \text{operation}(m) \rangle \rightarrow m$, for each match m.

4.4 Vectorization

Vector Pack. A pack is a tuple $\langle v, [m_1,\ldots,m_k]\rangle$, where v is a vector instruction with k output lanes, and $m_1,\ldots,m_k$ are a list of matches whose live-outs are independent.

- VeGen models vector loads and stores as two special kinds of packs, whose memory addresses must be contiguous.

Vector Operand. Vector Packs have vector operands, represented as a list of IR values. let $p=\langle v, [m_1,\ldots,m_k]\rangle$ be a vector pack, then $\text{operand}_i(p)=[x_1,\ldots,x_n]$; where $x_j \in \bigcup_k \text{live}_{\text{in}}(m_k)$ is one of the live-ins of the matches that should bind to the j'th lane of the i'th operand of the vector instruction v.

Don't-Care Lanes. Some instructions don't use all of their input lanes. Each element of a vector operand (i.e., $\text{operand}_i(.)$) therefore takes the value of either a scalar IR value (from the input program) or don't-care.

Producing a Vector Operand. A pack p produces a vector operand x if they have the same size (i.e., $|\text{values}(p)|=|x|$) and, for every lane i, x_i is either $\text{values}(p)_i$ or don't-care. VeGen uses a separate routine to enumerate producer packs that are vector loads, which can be done efficiently because *only contiguous loads can be packed together*.

Dependence and Legality. A pack p_1 depends on another pack p_2 if there exists an instruction $i \in \text{values}(p_1)$ that depends on another instruction $j \in \text{values}(p_2)$. A set of packs are legal when there are no cycles in the dependence graph.

Vector Pack Selection. Vectorization reduces to finding a subset of the matches and combining them into legal vector packs.

4.5 Code Generation

Given a set of vector packs (and the input program), VeGen's code generator emits a vector program as a combination of:

1. the scalar instructions not convered by the packs,
2. the compute vector instructions corresponding to the packs, and
3. the data-movement vector instructions that follow from the dependence among the packs and scalars.

The code generation algorithm uses the target-specific functions $\text{operand}_i(.)$ generated from instruction semantics.

Scheduling.

Lowering.

5 Vector Pack Selection

Optimization Objective and Cost Model.

Pack Selection Heuristics.

6 Implementation