RV: A Unified Region Vectorizer for LLVM

Simon Moll / Saarland University / Saarland Informatics Campus

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Introduction

The Region Vectorizer provides a single, unified API to vectorize code regions.

- RV is a generalization of the Whole-Function Vectorizer
  R. Karrenberg, S. Hack, "Whole Function Vectorization" (CGO '11)

```
rv::VectorizationInfo vi;
// region set up
rv::Region R(xLoop);
vi.setVectorShape(xPhi,
        VectorShape::consecutive());

// Vectorization analysis
rv::analyze(R, vi, domTree, loopInfo);

// Control conversion
rv::linearize(R, vi, domTree, loopInfo);

// Vector IR generation
rv::vectorize(R, vi, domTree);
```

Applications

- **Outer-Loop Vectorizer** An "unroll-and-jam" vectorizer based on RV's analysis and transformations
- **`pragma omp simd`** Emit vector code for SIMD regions right from Clang
- **Vectorizer Cost Model** How much predication? Which memory accesses vectorize well?
- **Polly** Directly vectorize loops during Polly code generation
- **PIR** Parallel region vectorizer

`rv::Region` Region

A region can be a subset of the basic blocks in a function or an entire function (`omp declare simd`).

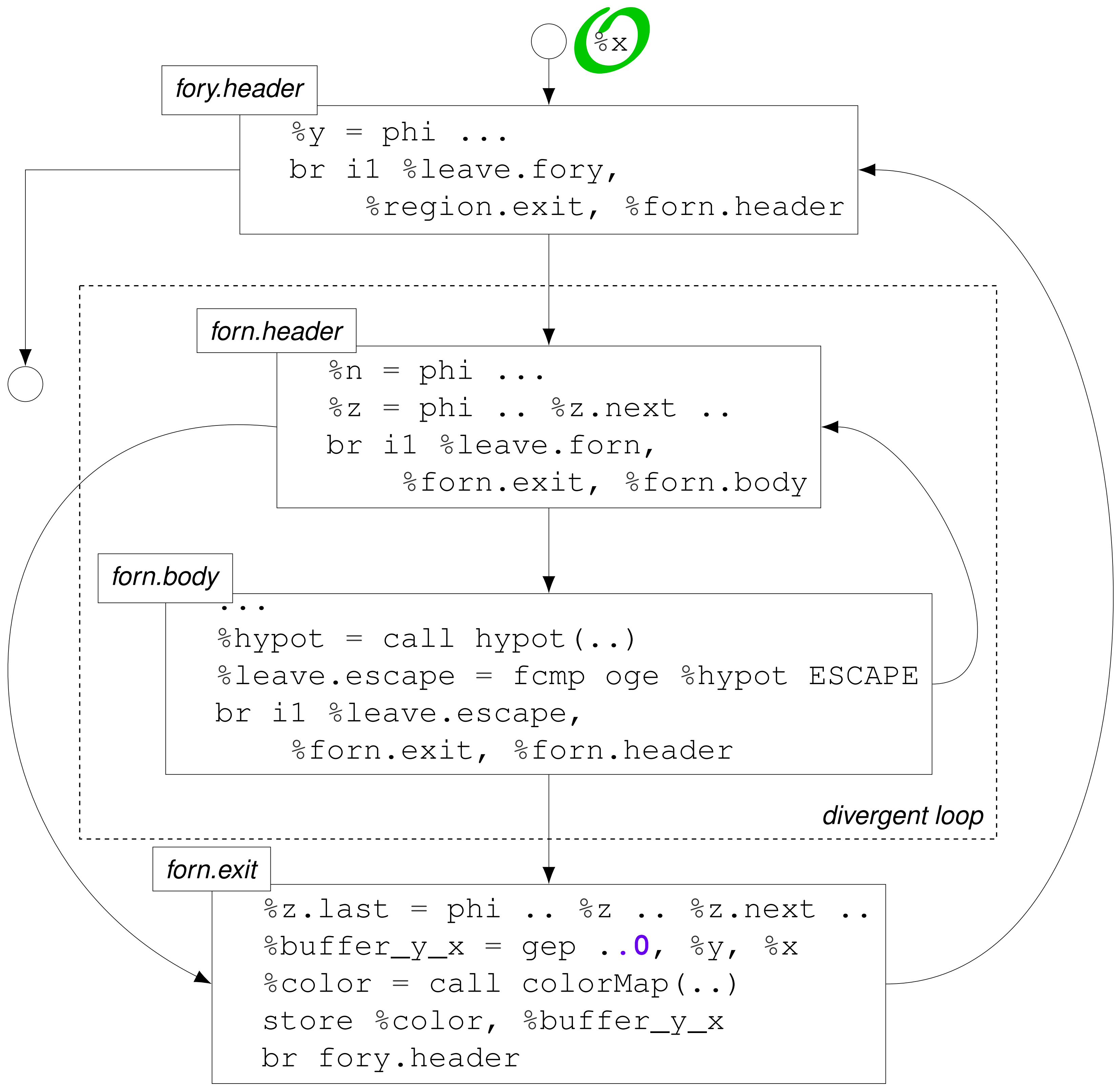
```
#pragma omp simd
for (int x = 0; x < width; ++x) {
  for (int y = 0; y < height; ++y) {
    complex<double> c = (startX+x*step) + (startY-y*step) * I;
    complex<double> z = 0.0;

    for (int n = 0; n < MAX_ITER; ++n) {
      z = z * z + c;
      if (hypot(z.real, z.imag) >= ESCAPE)
        break;
    }                                                   divergent loop
    buffer[y][x] = colorMap(z);
  }
}
```

```
#pragma omp declare simd
float min (float a, float b)
{
    if (a < b) return a; else return b;
}
```

↓

```
float min_v8 (<8 x float> a, <8 x float> b) {
  return select(a < b, a, b);
}
```

`rv::analyze` Vectorization Analysis



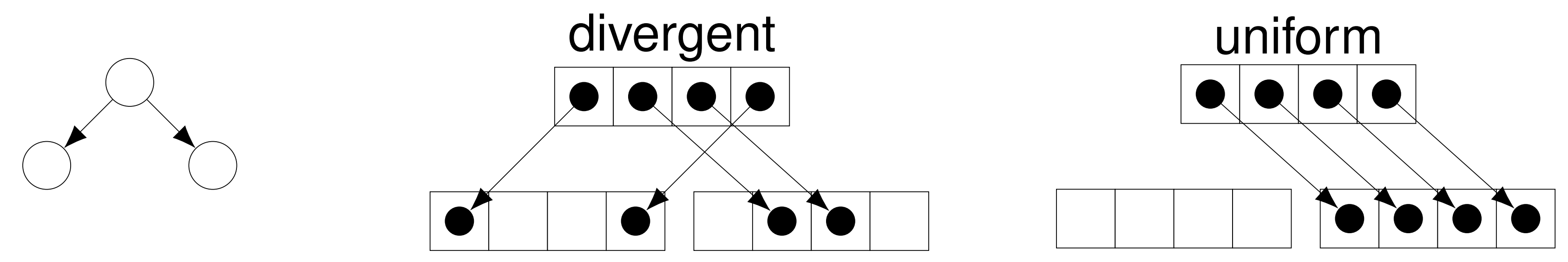
```
fory.header
  %y = phi ...
  br i1 %leave.fory,
      %region.exit, %forn.header

forn.header
  %n = phi ...
  %z = phi .. %z.next ..
  br i1 %leave.forn,
      %forn.exit, %forn.body

forn.body
  ...
  %hypot = call hypot(..)
  %leave.escape = fcmp oge %hypot ESCAPE
  br i1 %leave.escape,
      %forn.exit, %forn.header

forn.exit
  %z.last = phi .. %z .. %z.next ..
  %buffer_y_x = gep ..0, %y, %x
  %color = call colorMap(..)
  store %color, %buffer_y_x
  br fory.header
```

(***stride***, ***alignment***) or ⊤

`i64`	`%x`	0	1	2	3	4	5	6	7	. . .	**(1, 4)**	*(consecutive)*
`i64`	`%y, %n`	3	3	3	3	4	4	4	4	. . .	**(0, 1)**	*(uniform)*
`i1`	`%leave.escape`	1	1	0	0	0	0	1	1	. . .	⊤	*(varying)*

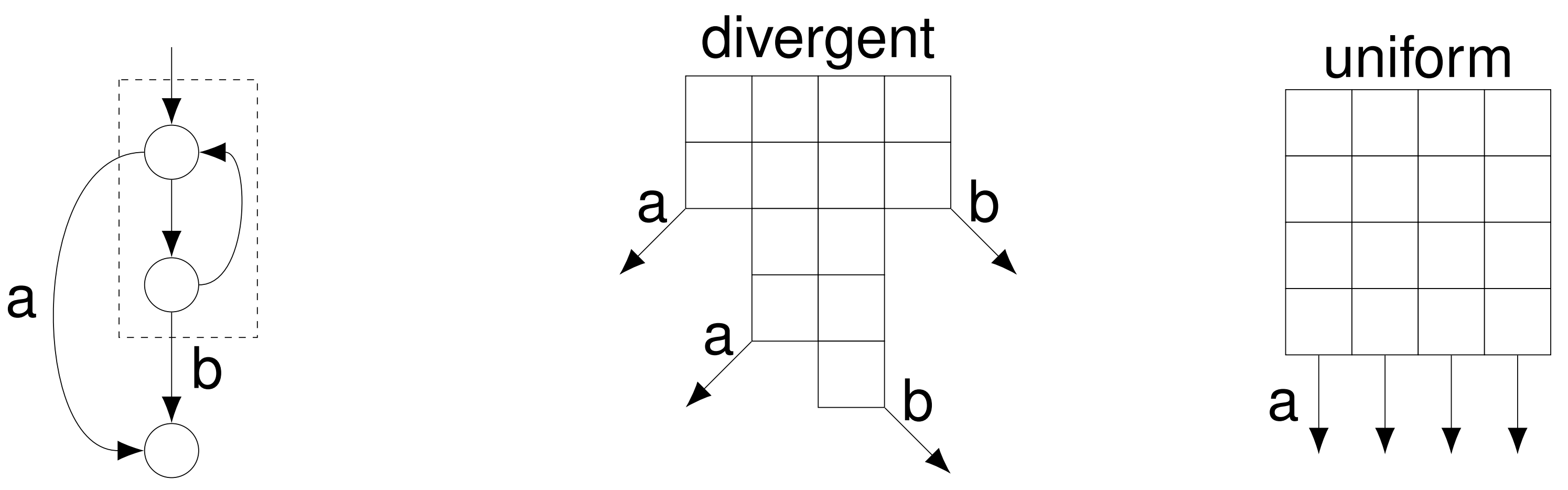
Branch Divergence

Which branches cause SIMD threads to diverge?



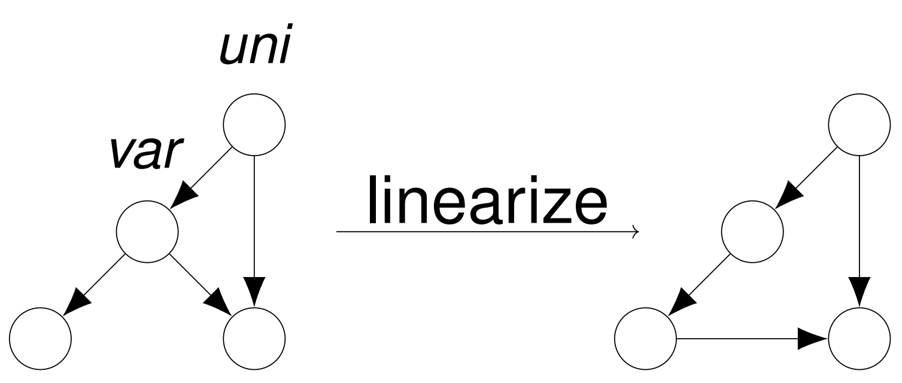
Loop Divergence

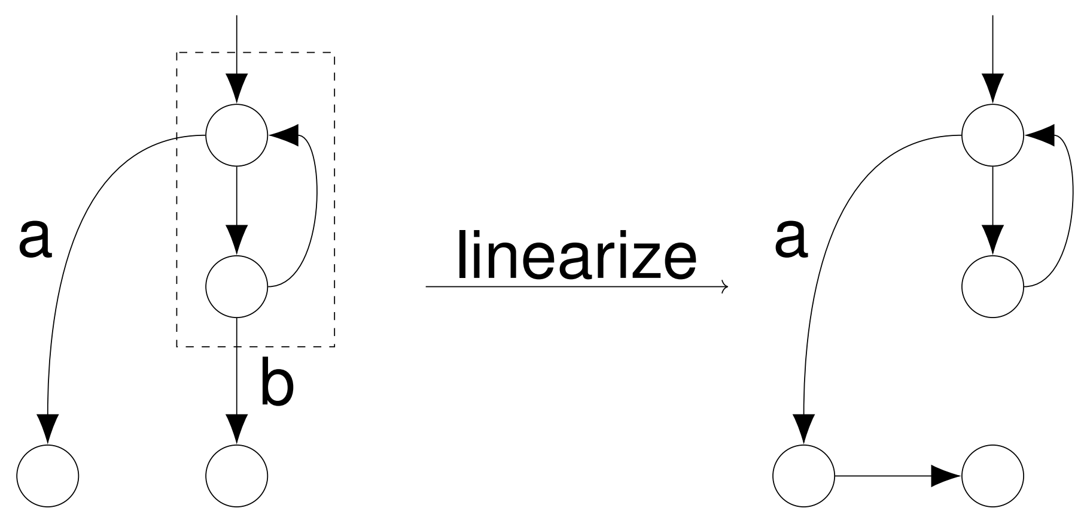
Which loops drop off SIMD threads at different exits?



`rv::linearize` Control Conversion

- Optimized linearization of divergent branches and loops (→ predication)
- Preserves uniform branches and loops
- Generates Predicated IR
  1. All branches are uniform
  2. Blocks may be predicated





Future Work

- BOSCC (skip predicated regions if no SIMD thread is active)
  J. Shin, "Introducing Control Flow into Vectorized Code" (PACT '07)
- Multi-dimensional Analysis
  C. Yount, "Vector Folding: Improving Stencil Performance via Multi-dimensional SIMD-vector Representation" (ICESS-CSS-HPCC '15)
- Vectorization of interleaved memory accesses

- Integration with Clang / LoopVectorizer / Polly
- Reductions

- Development available at GitHub
  `https://github.com/simoll/rv`

moll@cs.uni-saarland.de