RV: A Unified Region Vectorizer for LLVM

Simon Moll

Compiler Design Lab
Saarland Informatics Campus
Saarbrücken Germany

SAARLAND
UNIVERSITY

COMPUTER SCIENCE

```
#pragma omp simd
for (int y = 0; y < height; ++y) {
  for (int x = 0; x < width; ++x) {
    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;
    }
    buffer[y][x] = colorMap(z);
  }
}
```

```
#pragma omp simd
for (int y = 0; y < height; ++y) {
  for (int x = 0; x < width; ++x) {
    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;
    }
    buffer[y][x] = colorMap(z);
  }
}
```

```
#pragma omp simd
for (int y = 0; y < height; ++y) {
  for (int x = 0; x < width; ++x) {
    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;
    }
    buffer[y][x] = colorMap(z);
  }
}
```

LoopVectorizer can not handle outer loops.

```
#pragma omp simd
for (int y = 0; y < height; ++y) {
  for (int x = 0; x < width; ++x) {
    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;
    }
    buffer[y][x] = colorMap(z);
  }
}
```

LoopVectorizer can not handle outer loops.

→ **RV can vectorize it.**

rv::Region

```
#pragma omp simd
for (int y = 0; y < height; ++y) {
  for (int x = 0; x < width; ++x) {
    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;
    }
    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::Region

```
#pragma omp simd
for (int y = 0; y < height; ++y) {
  for (int x = 0; x < width; ++x) {
    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;
    }
    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::Region

```
#pragma omp simd
for (int y = 0; y < height; ++y) {
  for (int x = 0; x < width; ++x) {
    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;
    }
    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::Region

```
#pragma omp simd
for (int y = 0; y < height; ++y) {
  for (int x = 0; x < width; ++x) {
    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;
    }
    buffer[y][x] = colorMap(z);
  }
}
```

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

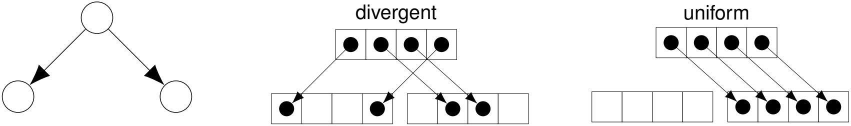
→

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

Unified API

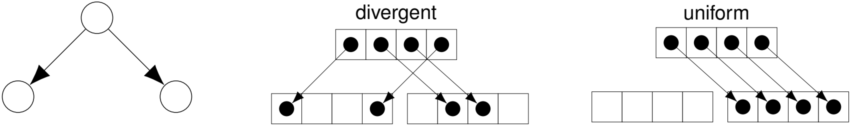
Unified API

- `rv::analyze(rv::Region& , ..)`



Unified API

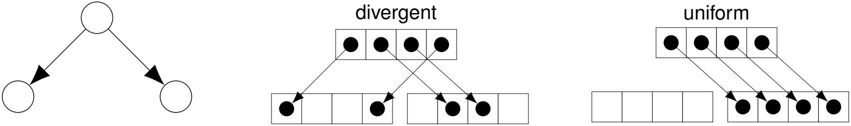
- `rv::analyze(rv::Region& , ..)`



- `rv::linearize`
  - *If-Conversion / Loop predication.*
  - *Preserves uniform control.*

Unified API

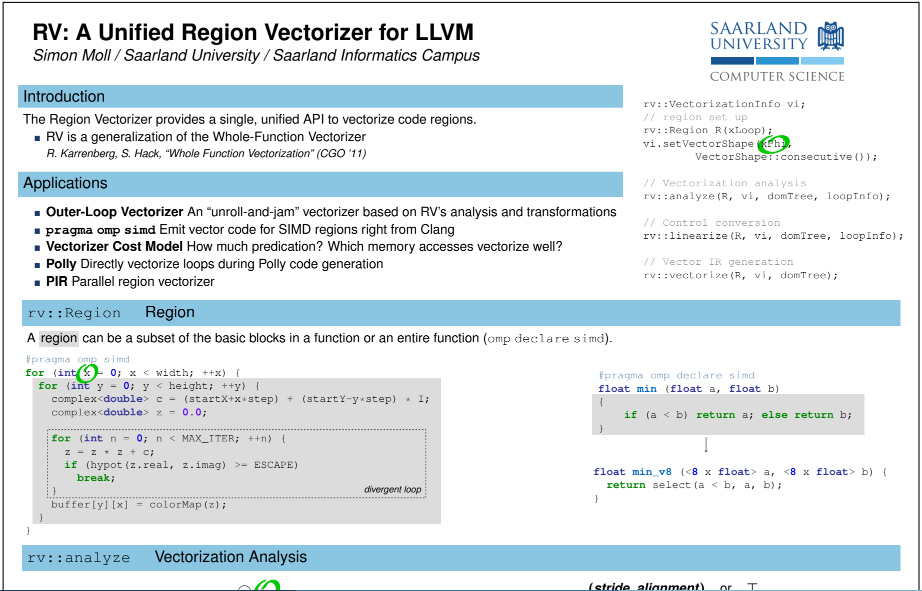
- `rv::analyze(rv::Region& , ..)`



- `rv::linearize`
  - *If-Conversion / Loop predication.*
  - *Preserves uniform control.*
- `rv::vectorize`

  Vector IR generation.

Poster



RV: A Unified Region Vectorizer for LLVM

Simon Moll / Saarland University / Saarland Informatics Campus

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)

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::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);
```

`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

(stride, alignment) or ⊤