VPlan + RV: A Proposal

Simon Moll and Sebastian Hack
October 18, 2017

http://compilers.cs.uni-saarland.de

Compiler Design Lab
Saarland University
Question
Have you come across a vectorizable loop that Clang could not vectorize?

```c
#pragma clang loop vectorize(assume_safety) 
vectorize_width(4)
for (int i = 0; i < m; i++) {
  double x = xs[i];
  double u0 = 0, u1 = 0, u2 = 0;
  for (int k = n; k >= 0; k--) {
    u2 = u1;
    u1 = u0;
    u0 = 2 * x * u1 - u2 + coeffs[k];
  }
  ys[i] = 0.5 * (coeffs[0] + u0 - u2);
}
```

Today, LLVM still won't (actually) vectorize the loop. → The Region Vectorizer can.

"[llvm-dev] autovectorization of outer loop", May 10, 2017
Question

Have you come across a vectorizable loop that Clang could not vectorize?

```c
#pragma clang loop vectorize(assume_safety) \ 
    vectorize_width(4)
for (int i=0; i<m; i++) {
    double x = xs[i];
    double u0=0, u1=0, u2=0;
    for (int k=n; k>=0; k--) {
        u2 = u1;
        u1 = u0;
        u0 = 2*x*u1-u2+coeffs[k];
    }
    ys[i] = 0.5*(coeffs[0]+u0-u2);
}
```

"[llvm-dev] autovectorization of outer loop", May 10, 2017
Question
Have you come across a vectorizable loop that Clang could not vectorize?

```
#pragma clang loop vectorize(assume_safety) \ strip
    vectorize_width(4)

for (int i=0; i<m; i++) {
    double x = xs[i];
    double u0=0, u1=0, u2=0;
    for (int k=n; k>=0; k--) {
        u2 = u1;
        u1 = u0;
        u0 = 2*x*u1-u2+coeffs[k];
    }
    ys[i] = 0.5*(coeffs[0]+u0-u2);
}
```

"'[llvm-dev] autovectorization of outer loop', May 10, 2017"


**Question**

Have you come across a vectorizable loop that Clang could not vectorize?

```c
#pragma clang loop vectorize(assume_safety) 
   vectorize_width(4)
for (int i=0; i<m; i++) {
  double x = xs[i];
  double u0=0, u1=0, u2=0;
  for (int k=n; k>=0; k--) {
    u2 = u1;
    u1 = u0;
    u0 = 2*x*u1-u2+coeffs[k];
  }
  ys[i] = 0.5*(coeffs[0]+u0-u2);
}
```

Today, LLVM still won’t (actually) vectorize the loop.
→ The Region Vectorizer can.

"[llvm-dev] autovectorization of outer loop", May 10, 2017
#pragma clang loop vectorize(assume_safety) \
    vectorize_width(4)

for (int i=0; i<m; i++){
    double x = xs[i];
    double u0=0, u1=0, u2=0;
    for (int k=n; k>=0; k--){
        u2 = u1;
        u1 = u0;
        u0 = 2*x*u1-u2+coeffs[k];
    }
    ys[i] = 0.5*(coeffs[0]+u0-u2);
}

clang++ -march=native -ffp-contract=fast \ 
    -fno-unroll-loops \ 
    -Xclang -load -Xclang libRV.so \ 
    -mllvm -rv-loopvec
#pragma clang loop vectorize(assume_safety) \
    vectorize_width(4)
for (int i=0; i<m; i++){
    double x = xs[i];
    double u0=0, u1=0, u2=0;
    for (int k=n; k>=0; k--){
        u2 = u1;
        u1 = u0;
        u0 = 2*x*u1-u2+coeffs[k];
    }
    ys[i] = 0.5*(coeffs[0]+u0-u2);
}

clang++ -march=native -ffp-contract=fast \
    -fno-unroll-loops \
    -Xclang -load -Xclang libRV.so \
    -mllvm -rv-loopvec

---

![Graph showing performance comparison between different compilers and optimization levels.](image)
LLVM’s LoopVectorizer

- only loops.
- only inner loops.
- only a single basic block (will if-convert all control flow).
- only very basic reduction patterns.
- complex, interdependent code base.
VPlan: Future of Vectorization in LLVM.

VPlan

Tentative plan to vectorize an (outer) loop without changing the IR.

\[ z = \frac{x}{y}; \]
\[ A[i] = z; \]
VPlan: Future of Vectorization in LLVM.

VPlan

Tentative plan to vectorize an (outer) loop without changing the IR.

```
z = x / y;
A[i] = z;
```

```
width=4
```

```
z = x / y;
A[i:i+3] = z;
```

VPlan: Future of Vectorization in LLVM.

VPlan

Tentative plan to vectorize an (outer) loop without changing the IR.

\[ z = x / y; \]
\[ A[i] = z; \]

width=4

\[ z = x / y; \]
\[ A[i:i+3] = z; \]

width=8

if-convert

\[ z = x / y; \]
\[ A[i:i+7] = z; \]
VPlan: Future of Vectorization in LLVM.

VPlan
Tentative plan to vectorize an (outer) loop without changing the IR.

Vectorizing with the VPlan infrastructure
First, pick the best VPlan, then execute it.
VPlan Goals

- Infrastructure for Vectorization.
VPlan Goals

- Infrastructure for Vectorization.
- If it’s a SIMD transformation, implement it on top of VPlan.
VPlan Goals

- Infrastructure for Vectorization.
- If it’s a SIMD transformation, implement it on top of VPlan.
  Short term: *Outer-Loop Vectorization.*
VPlan Goals

- Infrastructure for Vectorization.
- If it’s a SIMD transformation, implement it on top of VPlan.

Short term: *Outer-Loop Vectorization.*
Mid term: SLP vectorization, ..
VPlan Goals

- Infrastructure for Vectorization.
- If it’s a SIMD transformation, implement it on top of VPlan.
  Short term: *Outer-Loop Vectorization*.
  Mid term: SLP vectorization, ..
- OpenMP 4.5 SIMD pragmas (#pragma omp declare simd).
  "Extending LoopVectorizer towards supporting OpenMP 4.5 SIMD and outer loop auto-vectorization", Hideki Saito [devmtg ’16]
- Infrastructure for Vectorization.
- If it’s a SIMD transformation, implement it on top of VPlan. Short term: *Outer-Loop Vectorization*. Mid term: SLP vectorization, ..
VPlan Goals

- Infrastructure for Vectorization.
- If it’s a SIMD transformation, implement it on top of VPlan.
  Short term: *Outer-Loop Vectorization*.
  Mid term: SLP vectorization, ..
- OpenMP 4.5 SIMD pragmas (#pragma omp declare simd).
  "Extending LoopVectorizer towards supporting OpenMP4.5 SIMD
  and outer loop auto-vectorization", Hideki Saito [devmtg ’16]
  \(\rightarrow\) requires *Whole-Function Vectorization*.

VPlan Status

Infrastructure setup. VPlan is based on LoopVectorizer and shares its limitations.
VPlan Goals

- Infrastructure for Vectorization.
- If it’s a SIMD transformation, implement it on top of VPlan.
  Short term: **Outer-Loop Vectorization**.
  Mid term: SLP vectorization, ..
- OpenMP 4.5 SIMD pragmas (#pragma omp declare simd).
  "Extending LoopVectorizer towards supporting OpenMP4.5 SIMD
  and outer loop auto-vectorization”, Hideki Saito [devmtg ’16]
  → requires **Whole-Function Vectorization**.

VPlan Status

Infrastructure setup. VPlan is based on LoopVectorizer and shares its limitations.

Already available in RV.
Whole-Function and Outer-Loop Vectorizer.

- **powerful** strong analyses & transformations.
- **robust** vectorize any control-flow.
- **simple** clean, modular API.

Available on github: https://github.com/cdl-saarland/rv
Whole-Function and Outer-Loop Vectorizer.

- **powerful** strong analyses & transformations.
- **robust** vectorize any control-flow.
- **simple** clean, modular API.

Available on github: https://github.com/cdl-saarland/rv
void body(double* A, i) {
    A[i] = 5.0;
}

for (int i=0; i<VF; ..) {
    A[i] = 5.0;
}
Wrap body in loop and Loop Vectorize. [VecClone Pass, D22792]

```c
void body(double* A, i) {
    A[i] = 5.0;
}
```

```c
for (int i=0; i<VF; ..) {
    A[i] = 5.0;
}
```
Wrap body in loop and Loop Vectorize. [VecClone Pass, D22792]

What about vector arguments?

```c
void body(double*A, i) {
    A[i] = 5.0;
}
```

```c
for (int i=0; i<VF; ..) {
    A[i] = 5.0;
}
```

foo(float v) → foo_SIMD(float8 v)
Wrap body in loop and Loop Vectorize. [VecClone Pass, D22792]

What about vector arguments?

foo(float v) → foo_SIMD(float8 v)

Outline and Whole-Function Vectorize.

```c
void body(double* A, i) {
    A[i] = 5.0;
}
```
Wrap body in loop and Loop Vectorize. [VecClone Pass, D22792]

**What about vector arguments?**

foo(float v) → foo_SIMD(float8 v)

Outline and Whole-Function Vectorize.

**How do you outline recurrences & reductions?**

for (int i=0; i<VF; ..) {
  A[i] = 5.0;
}

void body(double* A, i) {
  A[i] = 5.0;
}
Wrap body in loop and Loop Vectorize. [VecClone Pass, D22792]

*What about vector arguments?*

```c
void body(double* A, i) {
    A[i] = 5.0;
}
```

```c
for (int i=0; i<VF; ..) {
    A[i] = 5.0;
}
```

Outline and Whole-Function Vectorize.

*How do you outline recurrences & reductions?*

```c
for (..) { ..; a += A[i]; ..; }
```
Wrap body in loop and Loop Vectorize. [VecClone Pass, D22792]

What about vector arguments?

foo(float v) → foo_SIMD(float8 v)

void body(double* A, i) {
    A[i] = 5.0;
}

Region Vectorize instead!

for (int i=0; i<VF; ..) {
    A[i] = 5.0;
}

Outline and Whole-Function Vectorize.

How do you outline recurrences & reductions?

for (..) { ..; a += A[i]; ..; }

7
Wrap body in loop and Loop Vectorize. [VecClone Pass, D22792]

*What about vector arguments?*

```c
void body(double*A, i) {
  A[i] = 5.0;
}
```

```c
for (int i=0; i<VF; ..) {
  A[i] = 5.0;
}
```

*Region Vectorize instead!*

```
for (..) { ..; a += A[i]; ..; }
```

Outline and Whole-Function Vectorize.

*How do you outline recurrences & reductions?*
Wrap body in loop and Loop Vectorize. [VecClone Pass, D22792]

*What about vector arguments?*

```c
void body(double* A, i) {
    A[i] = 5.0;
}
```

```c
for (int i=0; i<VF; ..) {
    A[i] = 5.0;
}
```

*Region Vectorize instead!*

Outline and Whole-Function Vectorize.

*How do you outline recurrences & reductions?*

```c
for (..) { ..; a += A[i]; ..; }
```
single entry.

multi exit.
(non-divergent)

rv::Region
To RV, loop vectorization and whole-function vectorization are (almost) the same.

```cpp
for (int i = 0; i < n; ++i) {
    ...
    continue;
    ...
}

double func(..) {
    ..  return 42.0; ..
    ..  return v;
}
```
• Statically precise sync dependences.
• Conditional reductions and recurrences.
  for (int i = 0; ...) {
    if (B[i] < 0.0) {
      a += C[i];
    }
  }
• Wavefront intrinsics (any, all, shuffle, ..)
  for (int i = 0; ...) {
    if (rv_any(p(i))) {
      /* for all if p(i) for any */
    }
  }
• Conversion of divergent loops (think: Mandelbrot).
• SLEEF vector math (AVX2, AVX512, Advanced SIMD, ..)
RV architecture

- Statically precise sync dependences.

- Divergence Analysis
- Recurrence Analysis
- Alloca Opt.
- BOSCC Heuristics
- Partial Linearization
- Vector IR Generator

```c
• Statically precise sync dependences.

```
RV architecture

- Statically precise sync dependences.
- Conditional reductions and recurrences.

```c
for (int i = 0; ...) {
    if (B[i] < 0.0) { a += C[i]; }
}
```

- Wavefront intrinsics (any, all, shuffle, ..)
- Conversion of divergent loops (think: Mandelbrot).
- SLEEF vector math (AVX2, AVX512, Advanced SIMD, ..)
- Statically precise sync dependences.
- Conditional reductions and recurrences.

```c
for (int i = 0; ...) {
    if (B[i] < 0.0) { a += C[i]; }
}
```

- Wavefront intrinsics (any, all, shuffle, ..)

```c
for (int i = 0; ...) {
    if (rv_any(p(i))) { /* for all if p(i) for any */ } }
```
- Statically precise sync dependences.
- Conditional reductions and recurrences.

```c
for (int i = 0; ...)
    if (B[i] < 0.0) { a += C[i]; }
```

- Wavefront intrinsics (any, all, shuffle, ..)

```c
for (int i = 0; ...)
    if (rv_any(p(i))) { /* for all if p(i) for any */ }
```

- Conversion of divergent loops (think: Mandelbrot).
- Statically precise sync dependences.
- Conditional reductions and recurrences.

```c
for (int i = 0; ...) {
    if (B[i] < 0.0) { a += C[i]; }
}
```

- Wavefront intrinsics (any, all, shuffle, ..)

```c
for (int i = 0; ...) {
    if (rv_any(p(i))) { /* for all if p(i) for any */ }
}
```

- Conversion of divergent loops (think: Mandelbrot).
- SLEEF vector math (AVX2, AVX512, Advanced SIMD, ..)
- **Pragma driven**  Will only vectorize where applied.
- **No cost model.**  Will do exactly as ordered.
- **No questions asked.** Incomplete legality checks
  
  All loop iterations/function instances have to be independent.
Pragma driven Will only vectorize where applied.

No cost model. Will do exactly as ordered.

No questions asked. Incomplete legality checks
  All loop iterations/function instances have to be independent.

→ these missing parts are already in VPlan/LLVM
The Proposal: VPlan + RV

Extensible Vectorization Infrastructure.
The Proposal: VPlan + RV

Extensible Vectorization Infrastructure.

VPlan

RV
The Proposal: VPlan + RV

Extensible Vectorization Infrastructure.

- Cost Modelling
- Legality Checks
- Runtime Checks
- Automatic Vectorizer

VPlan

RV
The Proposal: VPlan + RV

Extensible Vectorization Infrastructure.

Cost Modelling
Legality Checks
Runtime Checks
Automatic Vectorizer

VPlan

Divergence Analysis
Whole-Function Vectorization
Partial Linearization
Data Layout Optimizations

RV
The Proposal: VPlan + RV

Extensible Vectorization Infrastructure.

Cost Modelling
Legality Checks
Runtime Checks
Automatic Vectorizer

VPlan

Divergence Analysis
Whole-Function Vectorization
Partial Linearization
Data Layout Optimizations

RV

Start here!
**Embree**: Manually vectorized raytracer by Intel.

**Rodent**: RV-vectorized raytracer.

(A. Pérard-Gayot, Computer Graphics Lab & Intel Visual Computing Institute, Saarland University.)

- 238 uniform branches, 32 if-converted branches, 24 vectorized functions with 24 loops.
Let’s upstream RV’s divergence analysis!

- Joint Effort with Intel to integrate *Divergence Analysis* of RV into VPlan.
- Design Goals
  - no regressions compared to current SCEV-based implementation.
  - precise sync dependences.
#pragma clang loop vectorize(assume_safety) \nvectorize_width (4)
for (int i=0; i<m;i++){
    double x = xs[i];
    double ... O3 icc rv0
}

VPlan: Future of Vectorization in LLVM.
VPlan
Tentative plan to vectorize an (outer) loop without changing the IR.


A[i] = z;

Vectorizing with the VPlan infrastructure
First, pick the best VPlan, then execute it.

void body (double *A, i) {
    A[i] = 5.0;
}
for (int i=0; i<VF; ..) {
    A[i] = 5.0;
}
Wrap body in loop and Loop Vectorize. [VecClone Pass, D22792]

What about vector arguments?

foo (float v) → foo_SIMD (float8 v)

Outline and Whole-Function Vectorize.

How do you outline recurrences & reductions?

for (..) { ..; a += A[i]; ..; }
✗
✗
Region Vectorize instead!

The Proposal: VPlan + RV
Extensible Vectorization Infrastructure.
Cost Modelling
Legality Checks
Runtime Checks
Automatic Vectorizer
Divergence Analysis
Whole-Function Vectorization
Partial Linearization
Data Layout Optimizations
VPlan RV
Start here!

Divergence Analysis Stress Test: Rodent
• Embree: Manually vectorized raytracer by Intel.
• Rodent: RV-vectorized raytracer.
(A. Pérard-Gayot, Computer Graphics Lab & Intel Visual Computing Institute, Saarland University.)

• 238 uniform branches,
32 if-converted branches,
24 vectorized functions with
24 vectorized loops.

0
4
8
12
16
20
24
MRays/s
Rodent
Embree
Clang 4.0, AO rays, BVH8, AVX2, i7 6700K
Conclusion

RV Team

Thorsten Klössner    Dominik Montada    Arsène Pérard-Gayot    Simon Moll
Conclusion

RV Team
Thorsten Klössner  Dominik Montada  Arsène Pérard-Gayot  Simon Moll

VPlan talk, today 4:20pm
Vectorizing Loops with VPlan – Current State and Next Steps, Ayal Zaks
Backup
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

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

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

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

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

rv::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;
        }
        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);
}
```
Introduction
The Region Vectorizer provides a single, unified API to vectorize code regions.

- RV is a generalization of the Whole-Function Vectorizer

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

```cpp
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);
```

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

```cpp
#pragma omp declare 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;
    }
    buffer[y][x] = colorMap(z);
  }
}
```

```cpp
float min(float a, float b) {
  if (a < b) return a;
  else return b;
}
```

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

rv::analyze  Vectorization Analysis

```cpp
rv::analyze
```

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 (unification)
Introduction

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

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.

```c
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);
```

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

```c
#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;
  }
  buffer[y][x] = colorMap(z);
}
}
```

```c
float min(float a, float b) {
  if (a < b) return a; else return b;
}
```

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**

Development available at GitHub

https://github.com/simoll/rv

http://compilers.cs.uni-saarland.de

moll@cs.uni-saarland.de