



INTRODUCING VPLAN TO THE LOOP VECTORIZER

Gil Rapaport and Ayal Zaks

Intel Corporation, Israel Development Center

March 27-28, 2017 European LLVM Developers Meeting

Saarland Informatics Campus, Saarbrücken, Germany

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Key Takeaways

1. VPlan is an ongoing incremental effort to upgrade Loop Vectorizer's infrastructure and extend its capabilities
2. This effort is underway: first step introduces VPlan, reroutes vectorization decisions through it; early patches committed
3. VPlan's coverage to be extended in multiple directions going forward

The Need for VPlan

- LLVM's Loop Vectorizer (LV) is used extensively to optimize a large class of innermost loops
- But adding advanced vectorization techniques to LV is hard
 - Recent improvements already struggle
 - *Keep predicated instructions in the same block* [[D26555](#)]
 - Upcoming improvements magnify the difficulty
 - *RFC: Extending LV to vectorize outerloops* [[llvm-dev](#)]
 - *Extending LoopVectorizer towards supporting OpenMP4.5 SIMD and outer loop auto-vectorization* [[LLVM US'16](#)]
 - *RV: A Unified Region Vectorizer for LLVM - now on github* [[llvm-dev](#)]
- LV could vectorize loops better, and vectorize more loops

Need to upgrade LV's infrastructure to extend its capabilities

LV's Current Design and Major Limitations

1. Legality

RT aliasing checks

Must be scalarized

Uniform values

Requires predication

Interleave groups

2. Cost Model

Interleave groups

Should be scalarized

Sink to predicated BB

```
// Notice: any optimization or new instruction that go  
// into the code below should be also be implemented in  
// the cost-model.
```

3. Transform

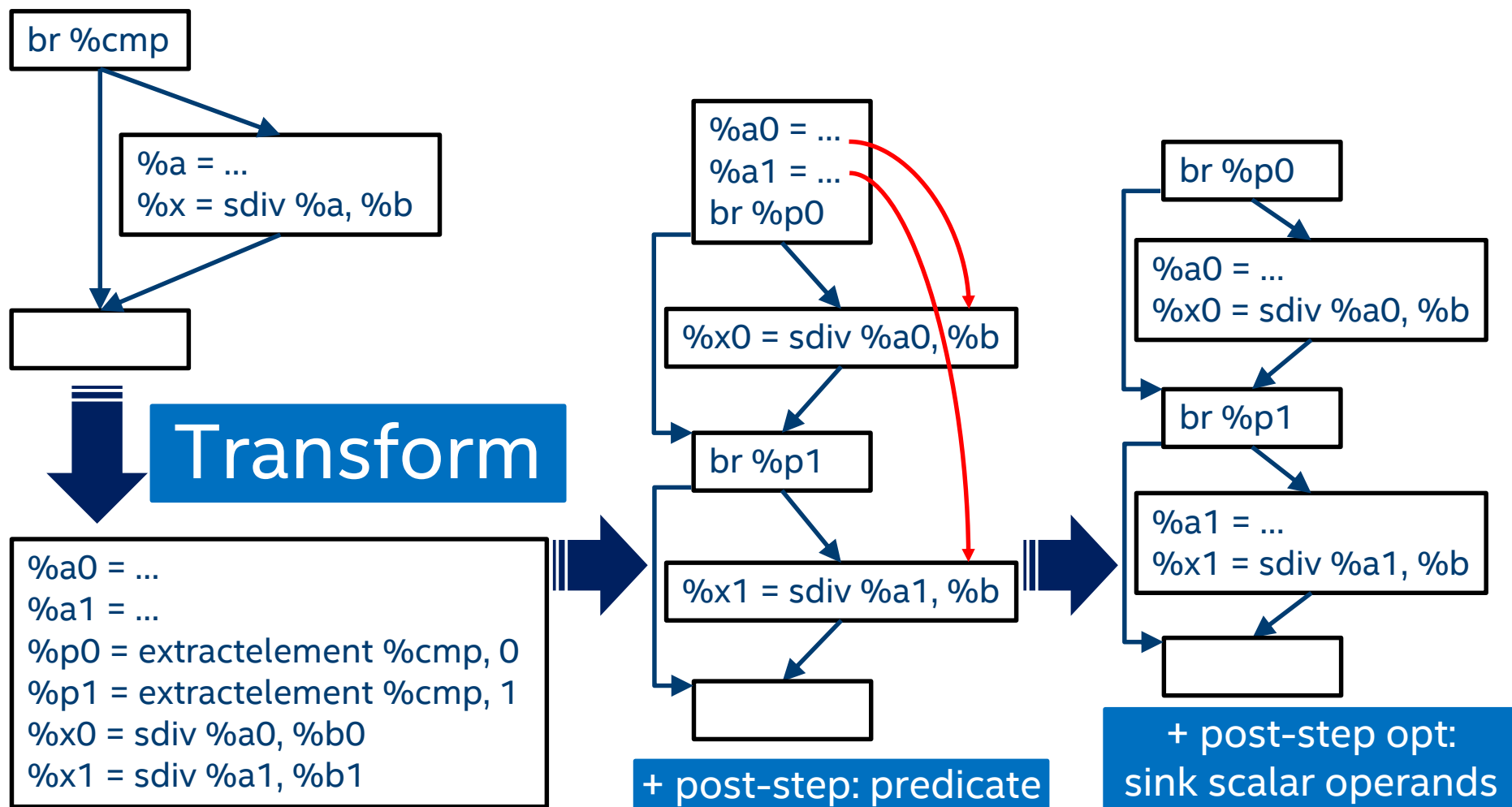
+ post-step: predicate

L3. Decisions recorded independently

L2. Hard to keep Cost aligned with Transform manually

L1. Output assumed to be a single basic block

Predication as a Post-Vectorization Step



Cost Model simulates Transform to calculate cost and optimize

VPlan Definitions

VPlan: a vectorized code candidate.
Uses a Hierarchical CFG (HCFG)

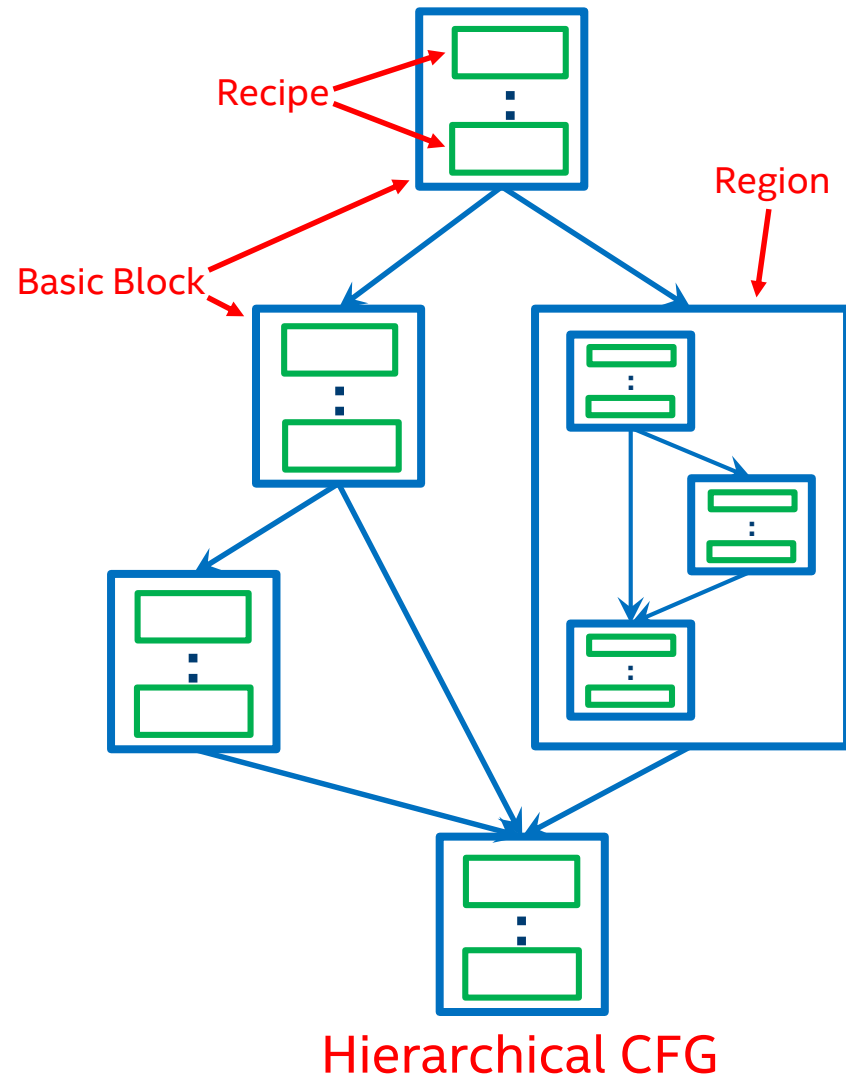
Block: an element of HCFG representing the control-flow of the vectorized code.

Basic Block: a leaf **Block**, contains a sequence of **Recipes**.

Region: an SESE subgraph of the HCFG.
Models vectorization semantics such as predication and replication.

Recipe: models a sequence of instructions to appear in the vectorized code. May refer to **Ingredients**.

Ingredient: an element of the original code, such as an instruction of the scalar loop.



VPlans calculate their cost and execute into IR

Recipe Example 1: Widening One-by-One

Source Code

```
void foo(int *a, int n, int *c) {  
    for (int i = 0; i < n; ++i)  
        a[i] = 3*c[2*i+1] + c[2*i];  
}
```



IR Before Vectorizer

```
for.body:  
...  
%0 = load i32, %arrayidx  
%mul1 = mul %0, 3  
%1 = load i32, %arrayidx3  
%add4 = add %mul1, %1  
store %add4, %arrayidx5  
...
```

VPlan for VF=4

```
...  
VECTORIZE RECIPE:  
%0 = load %arrayidx  
%mul1 = mul %0, 3  
%1 = load %arrayidx3  
%add4 = add %mul1, %1  
store %add4, %arrayidx5  
...
```

Ingredients

VPlan Execution

IR After Vectorizing for VF=4

```
vector.body:  
...  
%wmg = call @llvm.masked.gather.v4i32(%VecGep, ...)  
%50 = mul %wmg, <3,3,3,3>  
%wmg2 = call @llvm.masked.gather.v4i32(%VecGep2, ...)  
%84 = add %50, %wmg2  
store %84, %87  
...
```

VPlan strives to be lightweight by leveraging source IR

Recipe Example 2: Interleave Group

Source Code

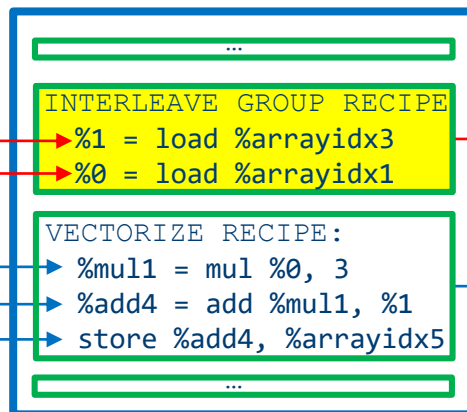
```
void foo(int *a, int n, int *c) {  
    for (int i = 0; i < n; ++i)  
        a[i] = 3*c[2*i+1] + c[2*i];  
}
```



IR Before Vectorizer

```
for.body:  
...  
%0 = load i32, %arrayidx  
%mul1 = mul %0, 3  
%1 = load i32, %arrayidx3  
%add4 = add %mul1, %1  
store %add4, %arrayidx5  
...
```

VPlan for VF=4

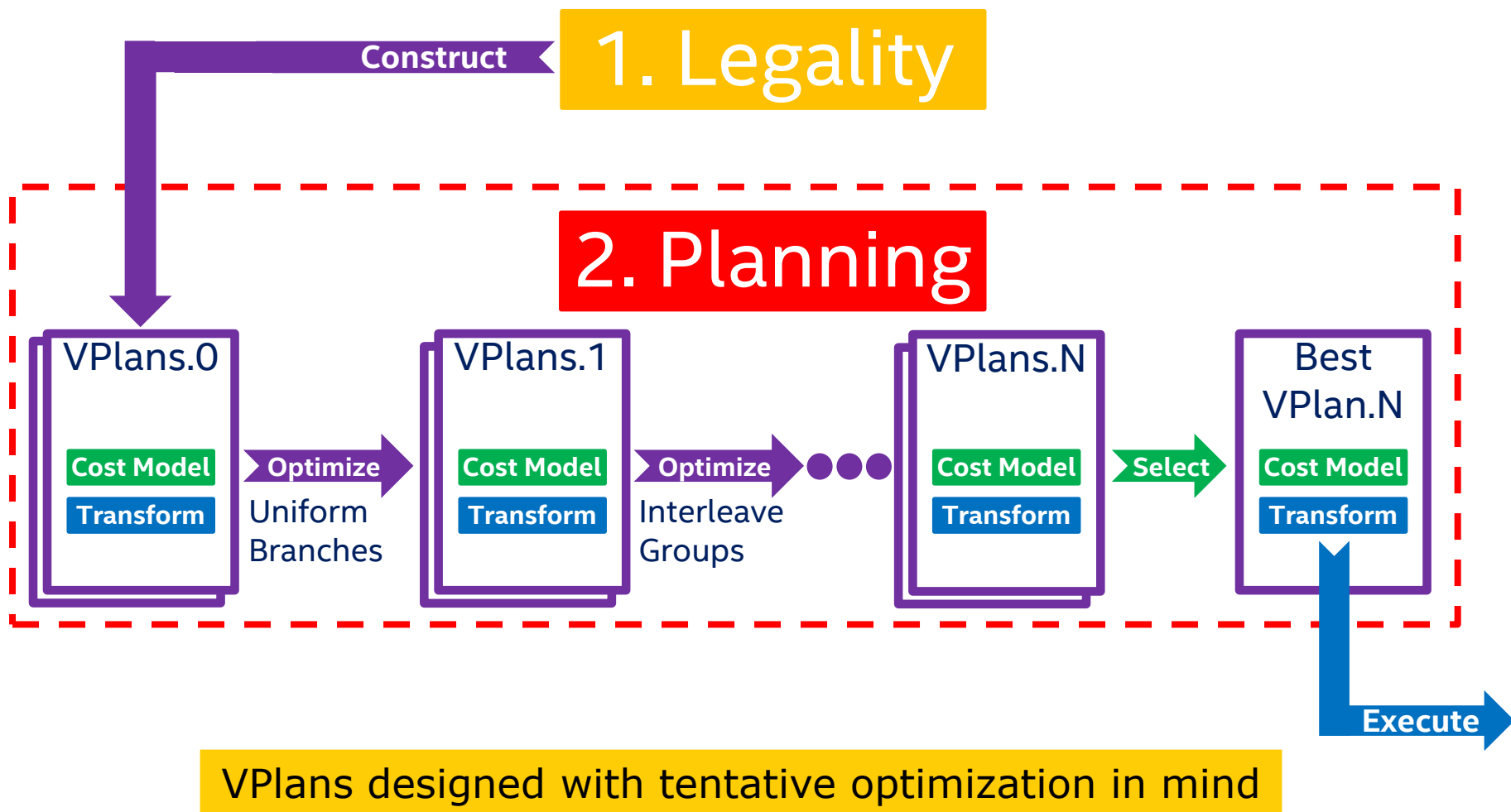


IR After Vectorizing for VF=4

```
vector.body:  
...  
%all = load <8 x i32>, %5  
%even = shufflevector %all, <0,2,4,6>  
%odd = shufflevector %all, <1,3,5,7>  
%6 = mul %odd, <3,3,3,3>  
%9 = add %6, %even  
store %9, %12  
...
```

Recipes capture simple and complex patterns as units of Cost

Modeling Decisions by Planning VPlans



How VPlan Addresses the Identified Limitations

LV's current limitation (recap)

1. Output assumed to be a single basic block
2. Hard to keep Cost aligned with Transform manually
3. Decisions recorded independently

LV with VPlan

1. Full control-flow is modelled explicitly
2. Single model of vectorized code simplifies and aligns both Cost and Transform
3. Single model represents a vectorized code candidate to manifest vectorization decisions explicitly

INTRODUCING VPLAN – CURRENT STATUS

Introducing VPlan by Refactoring Transform

LV's current design (recap)

1. Legality

2. Cost Model

3. Transform

+ post-step: predicate

LV with VPlan firstly introduced

1. Legality

2. Cost Model

Interleave
groups
Should be
scalarized

VPlans.0

Transform

Optimize

Sink
Scalar
Operands

VPlans.1

Transform

Select

Best
VPlan.1

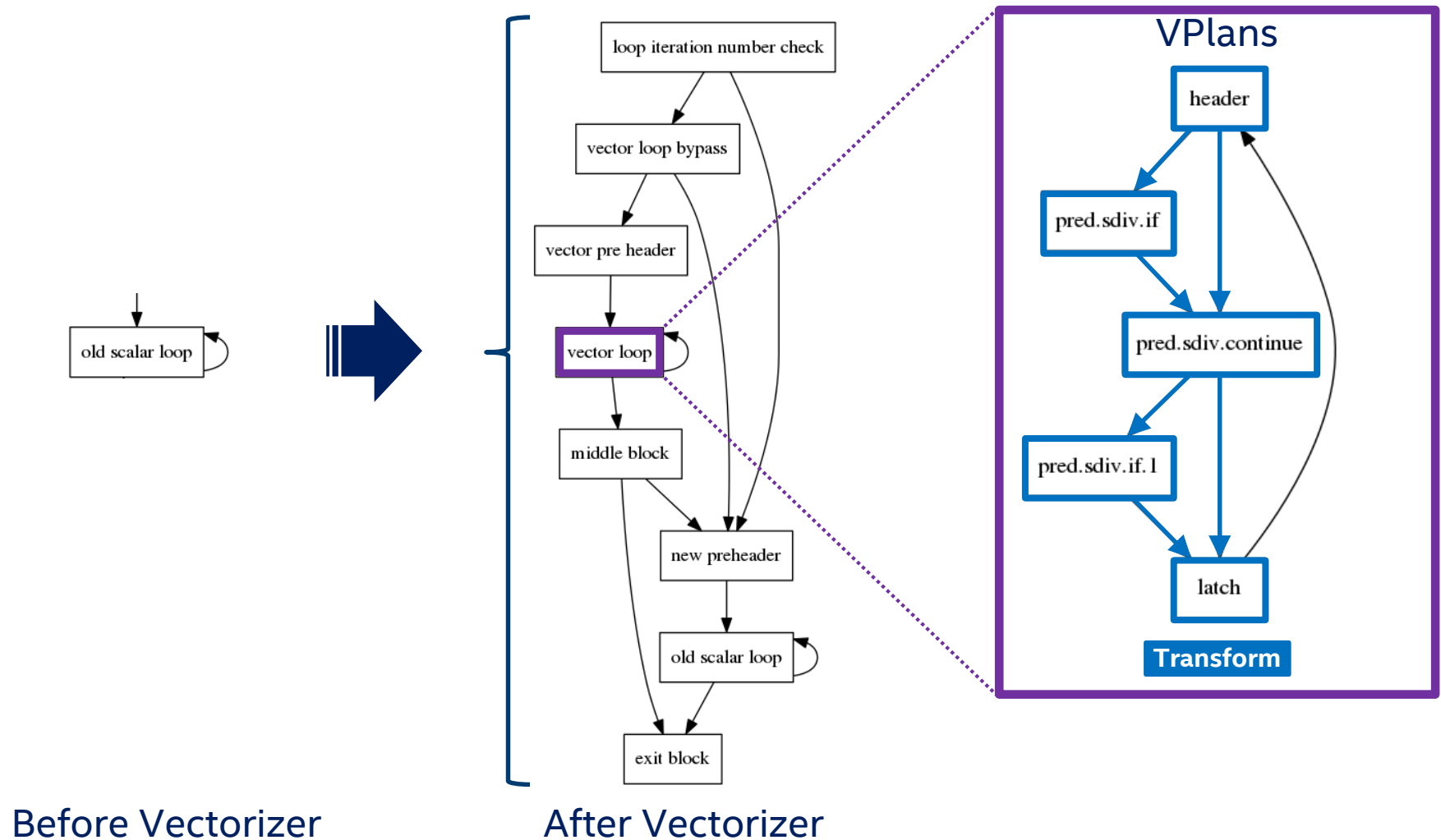
Transform

Execute

3. Planning

Construct

Introducing VPlan by Refactoring Transform, Cont'd



1st major step being committed gradually

A Concrete VPlan Example

Source Code

```
void foo(int *a, int b, int *c) {  
    for (int i = 0; i < 10000; ++i)  
        if (a[i] > 777)  
            a[i] = b - (c[100*i] * 7 + a[i]) / b;  
}
```



LLVM-IR Before Vectorizer

```
for.body:                ; preds = %for.inc, %entry  
    %i.015 = phi i32 [ 0, %entry ], [ %inc, %for.inc ]  
    %arrayidx = getelementptr inbounds i32, i32* %a, i32 %i.015  
    %0 = load i32, i32* %arrayidx, align 4  
    %cmp1 = icmp sgt i32 %0, 777  
    br i1 %cmp1, label %if.then, label %for.inc  
  
if.then:                 ; preds = %for.body  
    %mul = mul nuw nsw i32 %i.015, 100  
    %arrayidx2 = getelementptr inbounds i32, i32* %c, i32 %mul  
    %1 = load i32, i32* %arrayidx2, align 4  
    %mul3 = mul nsw i32 %1, 7  
    %add = add nsw i32 %mul3, %0  
    %div = sdiv i32 %add, %b  
    %sub = sub nsw i32 %b, %div  
    store i32 %sub, i32* %arrayidx, align 4  
    br label %for.inc  
  
for.inc:                 ; preds = %for.body, %if.then  
    %inc = add nuw nsw i32 %i.015, 1  
    %exitcond = icmp eq i32 %inc, 10000  
    br i1 %exitcond, label %for.cond.cleanup, label %for.body
```



VPlan for VF={2,4,8}

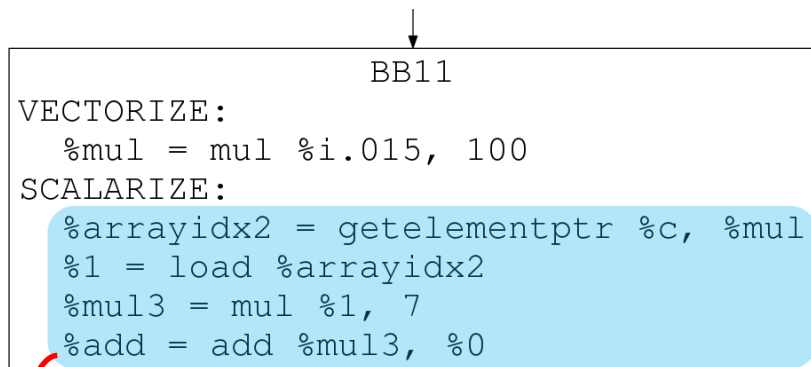
BB10
WIDEN INT INDUCTION (needs scalars):
 %i.015 = phi 0, %inc
BUILD SCALAR STEPS:
 %i.015 = phi 0, %inc
SCALARIZE:
 %arrayidx = getelementptr %a, %i.015
VECTORIZE:
 %0 = load %arrayidx
 %cmp1 = icmp %0, 777

BB11
VECTORIZE:
 %mul = mul %i.015, 100
SCALARIZE:
 %arrayidx2 = getelementptr %c, %mul
 %1 = load %arrayidx2
 %mul3 = mul %1, 7
 %add = add %mul3, %0

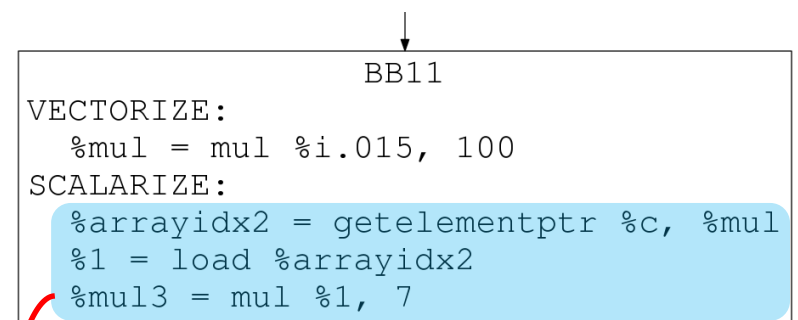
<xVFxUF> region13
BB14
EXTRACT MASK BIT:
 if.then
if.then
BB12
SCALARIZE:
 %div = sdiv %add, %b
!if.then
BB15
MERGE SCALARIZE BRANCH:
 %div = sdiv %add, %b

BB16
VECTORIZE:
 %sub = sub %b, %div
 store %sub, %arrayidx

VPlan-based sinkScalarOperands optimization (1/3)

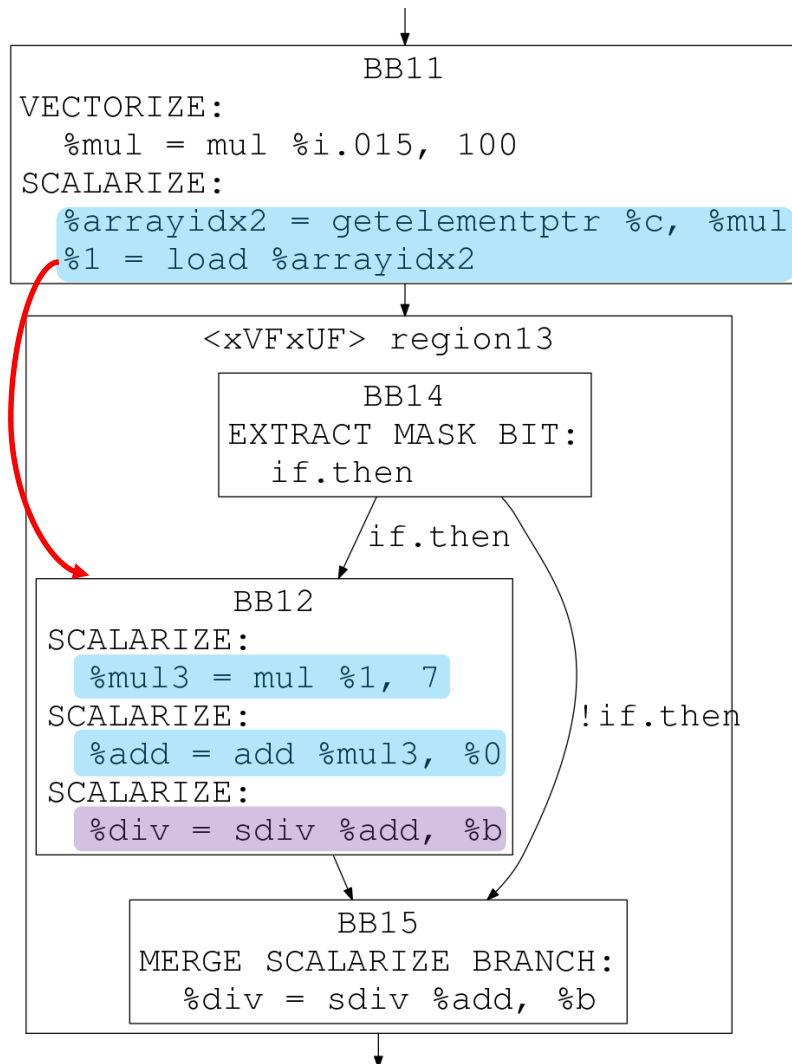


Initial State

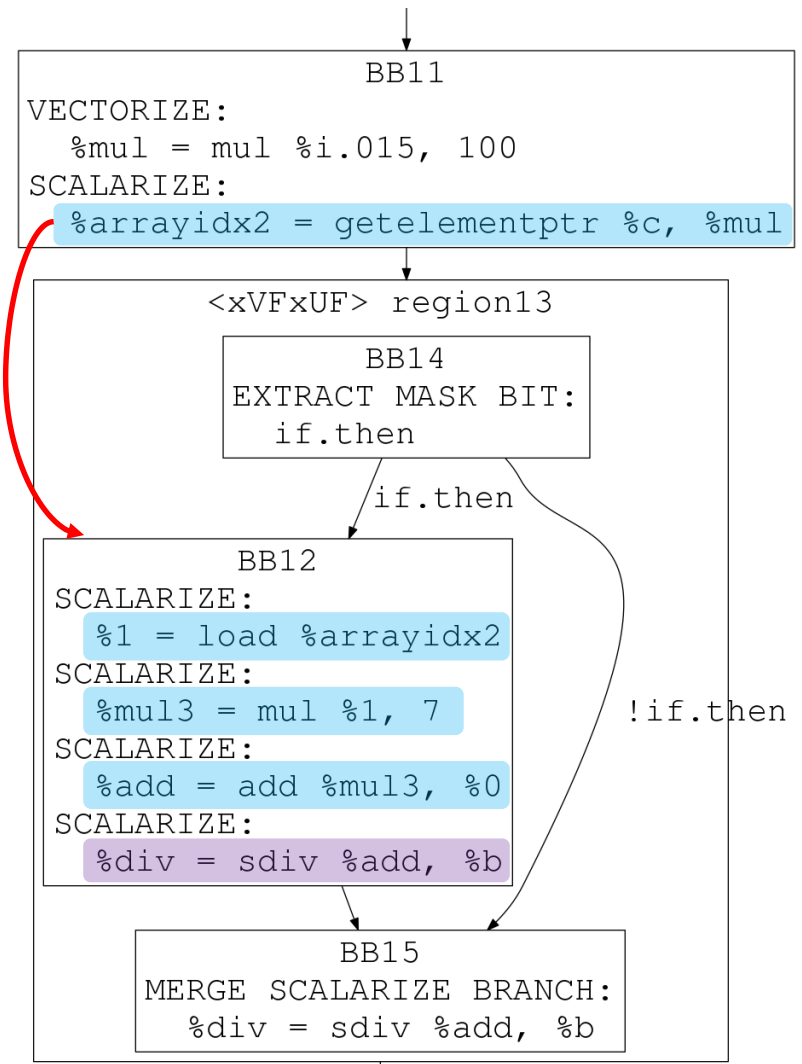


Sink {add}

VPlan-based sinkScalarOperands optimization (2/3)

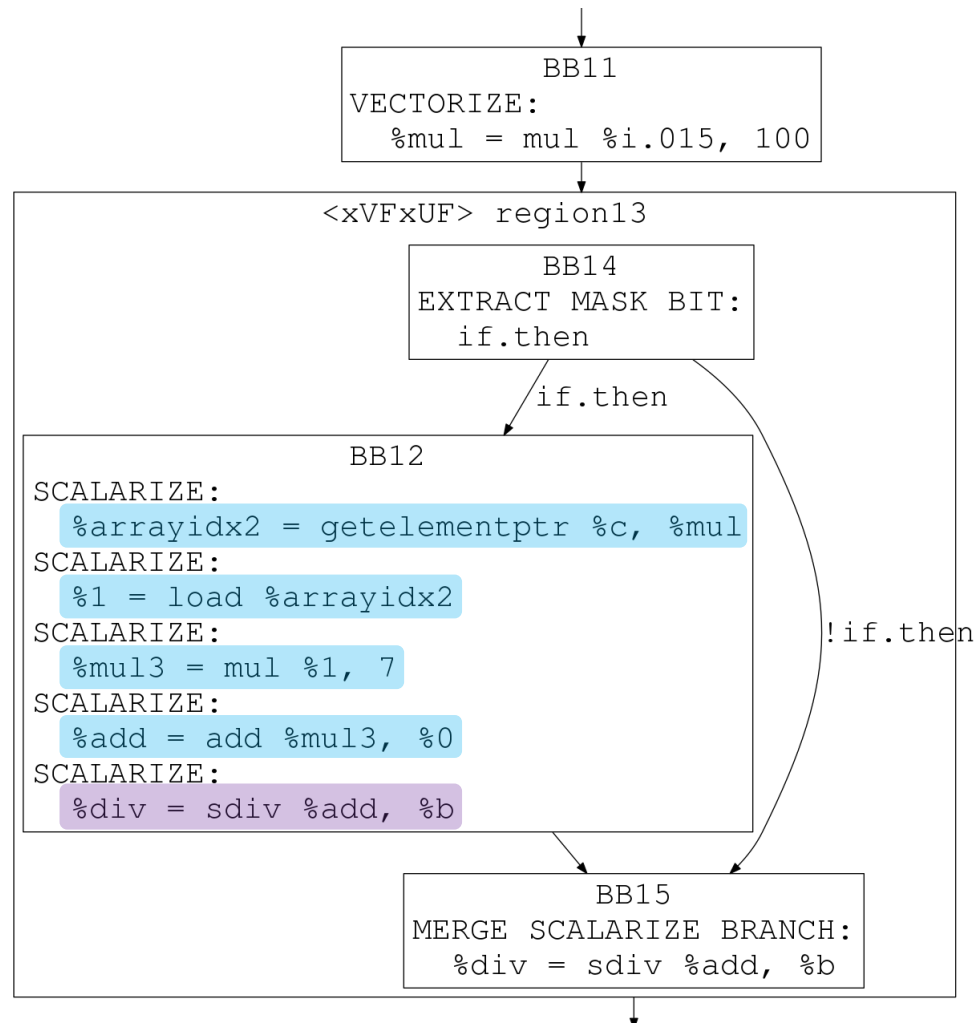


Sink {add, mul}



Sink {add, mul, load}

VPlan-based sinkScalarOperands optimization (3/3)



Sink{add, mul, load, gep}

Post-vectorization optimization modelled with VPlan

Key Takeaways

1. VPlan is an ongoing incremental effort to upgrade Loop Vectorizer's infrastructure and extend its capabilities
2. This effort is underway: first step introduces VPlan, reroutes vectorization decisions through it; early patches committed
3. VPlan's coverage to be extended in multiple directions going forward

