VECTORIZING LOOPS WITH VPLAN – CURRENT STATE AND NEXT STEPS

Ayal Zaks and Gil Rapaport, Vectorization Team, Intel Corporation

October 18th, 2017 US LLVM Developers' Meeting, San Jose, CA

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 © 2017, 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

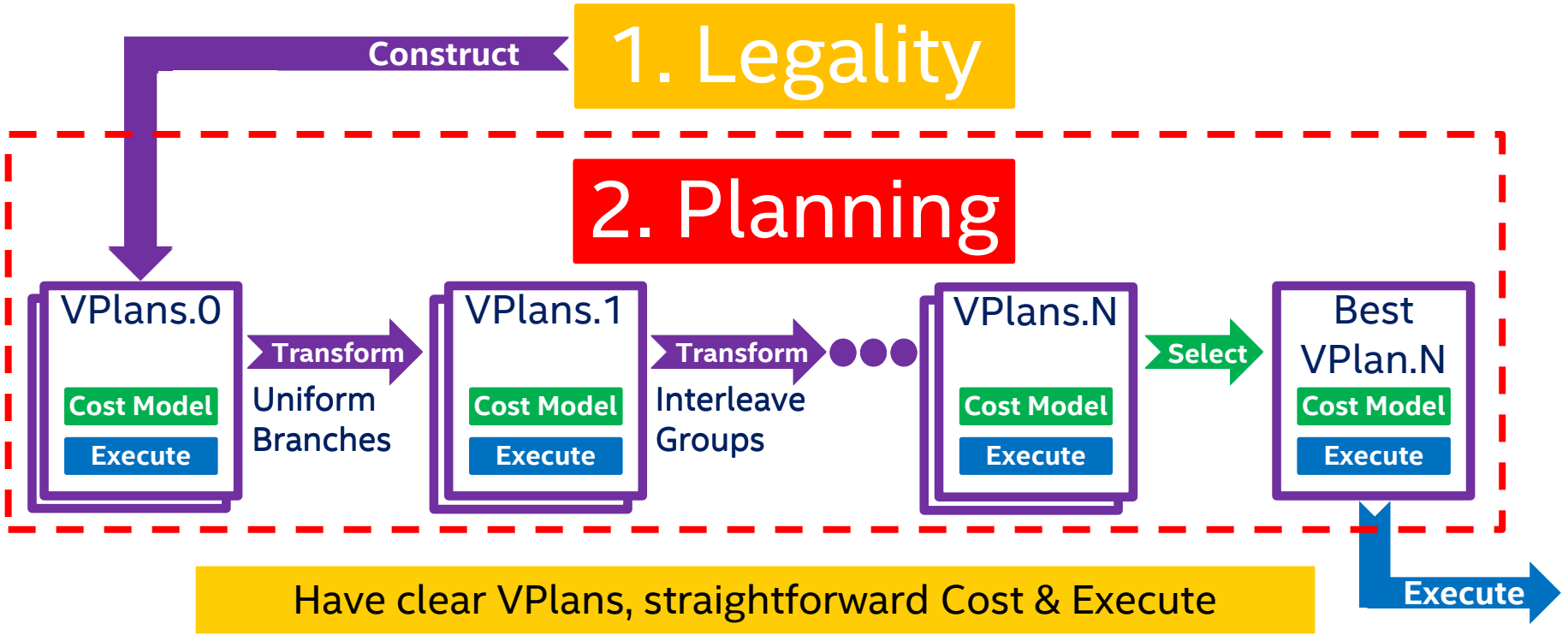
Key Takeaways

A. Current State: 1st step introducing VPlan to Loop Vectorizer – committed
   1. Records vectorization decisions in VPlan
   2. Drives vector code generation by executing a VPlan

B. Going Forward: shift vectorization process to be VPlan-based
   1. Refine the model, include masking and break Recipes into VPInstructions
   2. Carry out decisions based on VPlan, in addition to recording them
   3. Make decisions based on VPlan, including legal and cost-based analyses

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

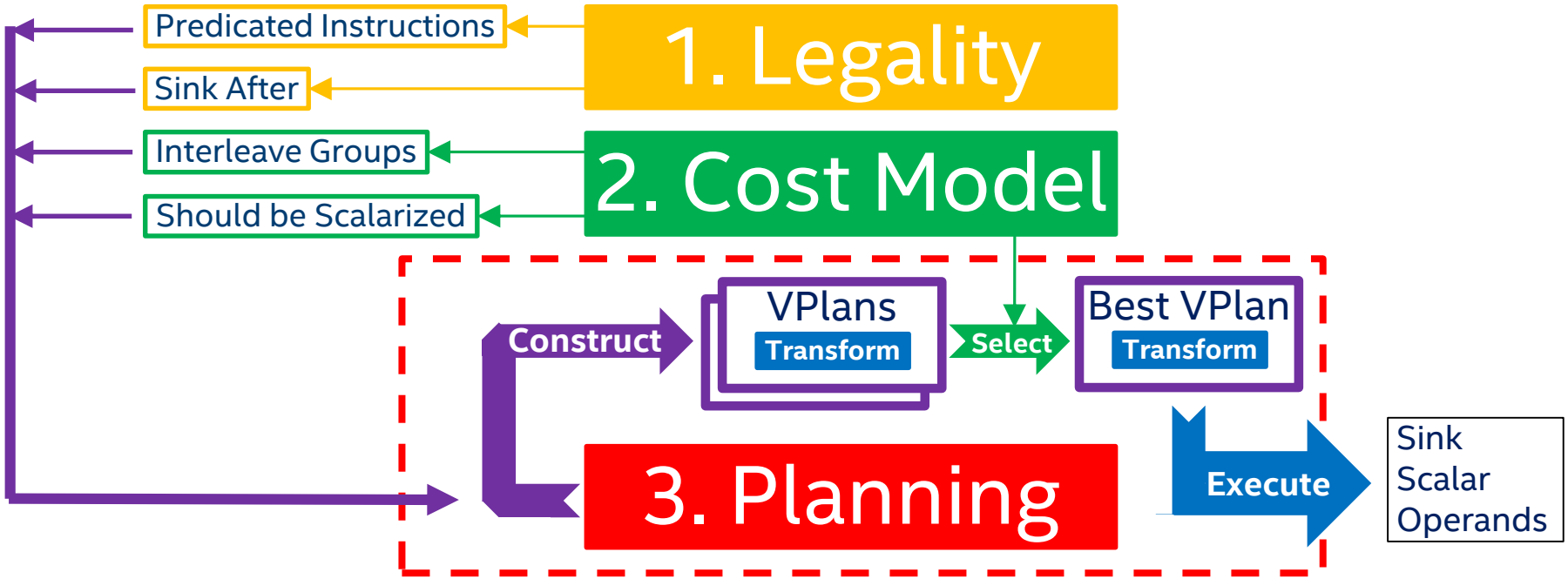
Recap: Loop Vectorization Plan



Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

A. CURRENT STATE OF VPLAN

1st Step Committed: VPlan Refactors Transform



Decisions taken up-front, during execute, or as post-pass

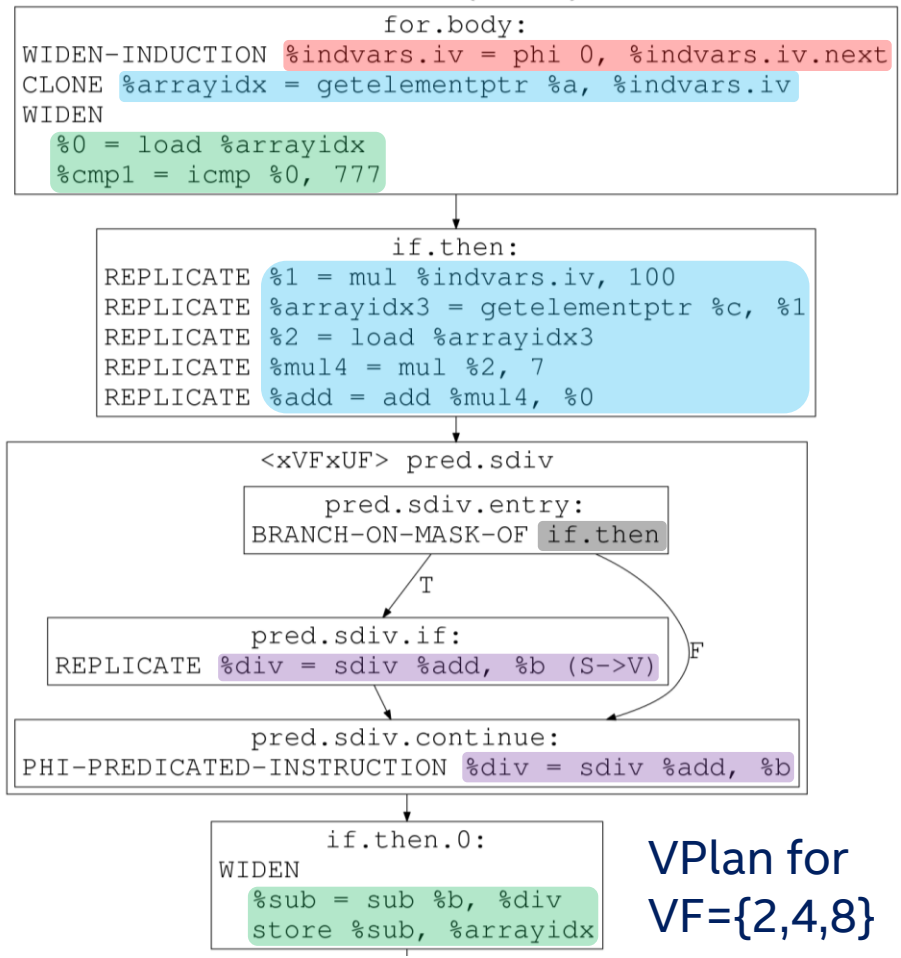
Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

VPlan Model: Current State

```
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:
 %indvars.iv = phi i64 [ 0, %entry ], [ %indvars.iv.next, %for.i
 %arrayidx = getelementptr inbounds i32, i32* %a, i64 %indvars.i
 %0 = load i32, i32* %arrayidx, align 4
 %cmp1 = icmp sgt i32 %0, 777
 br i1 %cmp1, label %if.then, label %for.inc
if.then:
 %1 = mul nuw nsw i64 %indvars.iv, 100
 %arrayidx3 = getelementptr inbounds i32, i32* %c, i64 %1
 %2 = load i32, i32* %arrayidx3, align 4
 %mul4 = mul nsw i32 %2, 7
 %add = add nsw i32 %mul4, %0
 %div = sdiv i32 %add, %b
 %sub = sub nsw i32 %b, %div
 store i32 %sub, i32* %arrayidx, align 4
 br label %for.inc
```



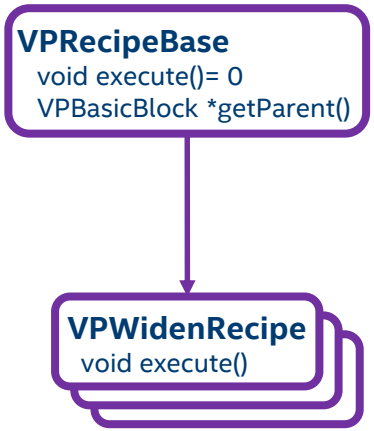
VPlan for VF={2,4,8}

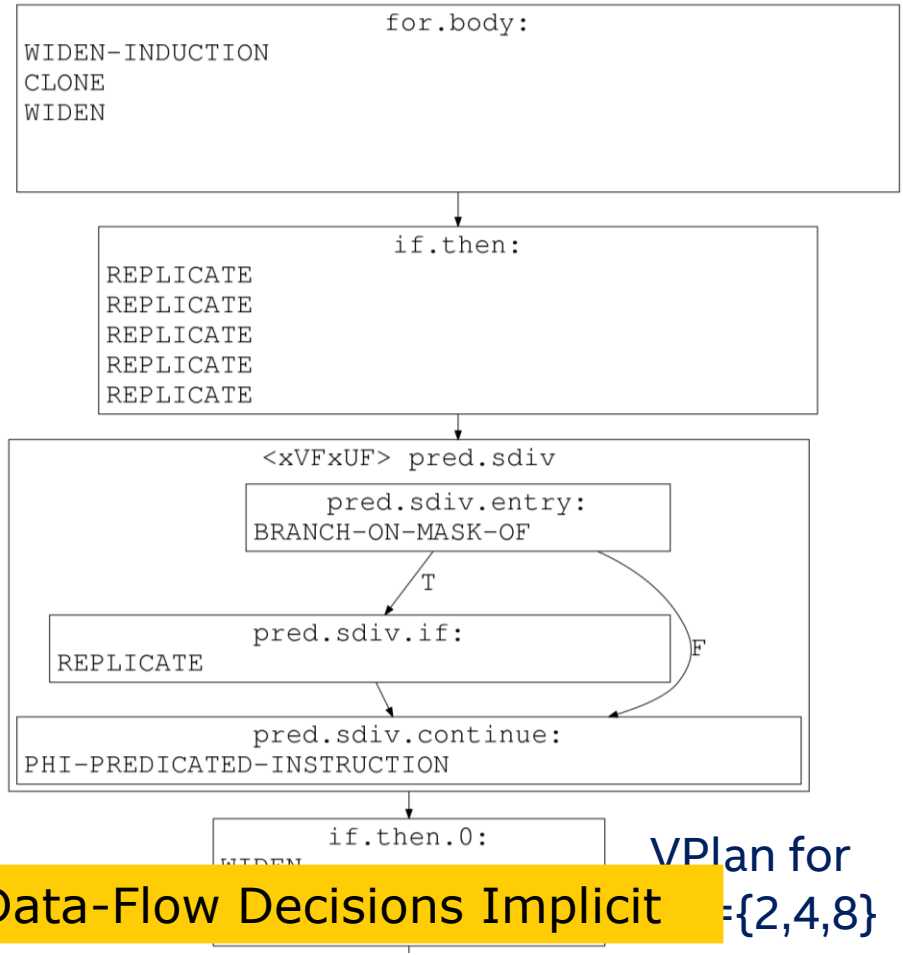
Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

VPlan Model: Current State

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

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





VPlan for ={2,4,8}

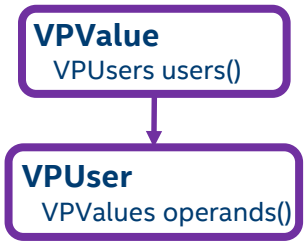
Control-Flow Decisions Explicit, Data-Flow Decisions Implicit

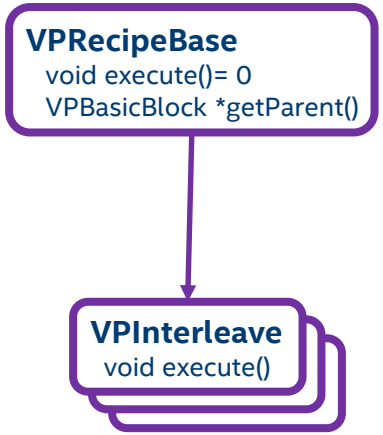
Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

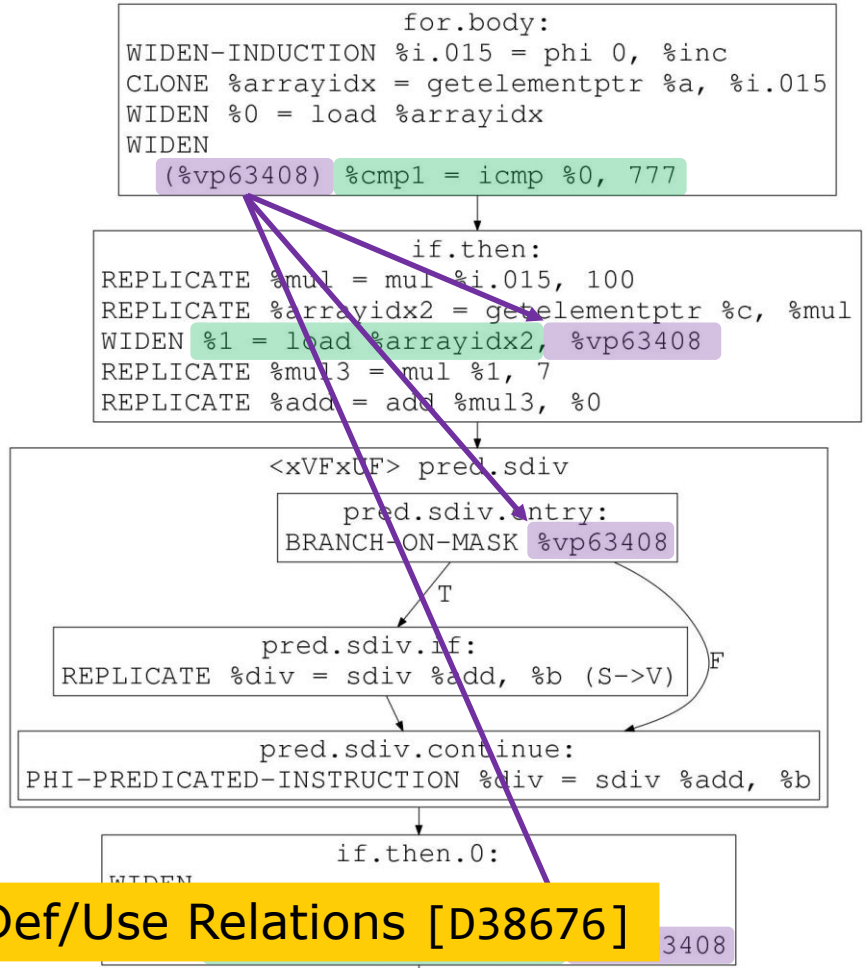
B.1. MODEL MASKING AND INSTRUCTIONS

VPlan Model: Next Step

```
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;
}
```





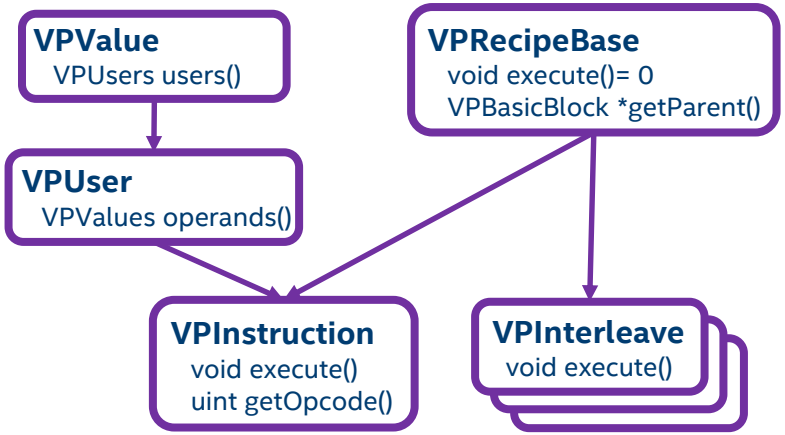


Model Masking in VPlan using Def/Use Relations [D38676]

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

VPlan Model: Next Step (cont'd)

```
void foo(int* a, int b, int* c) {
  for (int i = 0; i < 10000; ++i)
    if (a[i] > 777) {
      c[i] = b;
      if (a[i] > 888)
        a[i] = b;
    }
}
```



VPlan for VF={2,4,8,16}

```
                    for.body:
WIDEN-INDUCTION %i.017 = phi 0, %inc
CLONE %arrayidx = getelementptr %a, %i.017
WIDEN %0 = load %arrayidx
WIDEN
  (%vp27696) %cmp1 = icmp %0, 777
```

```
                    if.then:
CLONE %arrayidx2 = getelementptr %c, %i.017
WIDEN store %b, %arrayidx2, %vp27696
WIDEN %1 = load %arrayidx
WIDEN
  (%vp30784) %cmp4 = icmp %1, 888
```

```
                    if.then5:
EMIT %vp58664 = and %vp30784 %vp27696
WIDEN store %b, %arrayidx, %vp58664
```

VPInstruction: Instruction-level Modeling in VPlan [D38676]

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

B.2. FROM RECORDING DECISIONS TO CARRYING THEM OUT

Taking Decision (1/4): Interleave Groups

```
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

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

VPlan for VF=4

```
VPInterleaveRecipe:
  %1 = load %arrayidx3
  %0 = load %arrayidx1

VPWidenRecipe:
  %mul1 = mul %0, 3
  %add4 = add %mul1, %1
  store %add4, %arrayidx5
```

Ingredients

VPlan Execution

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
  …
```

Effectively hoists `load %1` to join `load %0`

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Taking Decision (2/4): Unravel 1st Order Recurrence

```
void sink_after(short *a, int *b, int n) {
  for (int i = 0; i < n; ++i)
    b[i] = (a[i] * a[i+1]);
}
```

IR Before Vectorizer

```
foo.body:
  %iv = phi i64 [ 0, %entry ], [ %iv.next, %for.body ]
  %0 = phi i16 [ %.pre, %entry ], [ %1, %for.body ]
  %conv = sext i16 %0 to i32
  %iv.next = add nuw nsw i64 %iv, 1
  %arrayidx2 = getelementptr i16, i16* %a, i64 %iv.next
  %1 = load i16, i16* %arrayidx2
  %conv3 = sext i16 %1 to i32
  %mul = mul nsw i32 %conv3, %conv
  %arrayidx5 = getelementptr i32, i32* %b, i64 %iv
  store i32 %mul, i32* %arrayidx5
  %exitcond = icmp eq i64 %indvars.iv.next, %n
  br i1 %exitcond, label %for.end, label %for.body
```

IR After Vectorizer

```
vector.body
  %iv = phi i64 [ 0, %vec.ph ], [ %iv.next, %vec.body ]
  %recur = phi <4 x i16> [ %recur.init, %vec.ph ],
                         [ %wide.load, %vec.body ]
  …
  %3 = getelementptr inbounds i16, i16* %a, i64 %2
  %wide.load = load <4 x i16>, <4 x i16>* %5, align 2
  %6 = shufflevector <4 x i16> %recur,
                     <4 x i16> %wide.load,
                     <4 x i32> <3, 4, 5, 6>
  %7 = sext <4 x i16> %6 to <4 x i32>
  %8 = sext <4 x i16> %wide.load to <4 x i32>
  %9 = mul nsw <4 x i32> %8, %7
```

Phase-ordering: first sink cast after load, then hoist interleave load [PR34743]

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Taking Decision (3/4): Predication

- Must convert divergent branches using masking
- Much more challenging for outer-loop vectorization
  - Earlier today: *VPlan + RV: A Proposal* by Simon Moll and Sebastian Hack
  - Last year's *Extending LoopVectorizer:* by Hideki Saito:

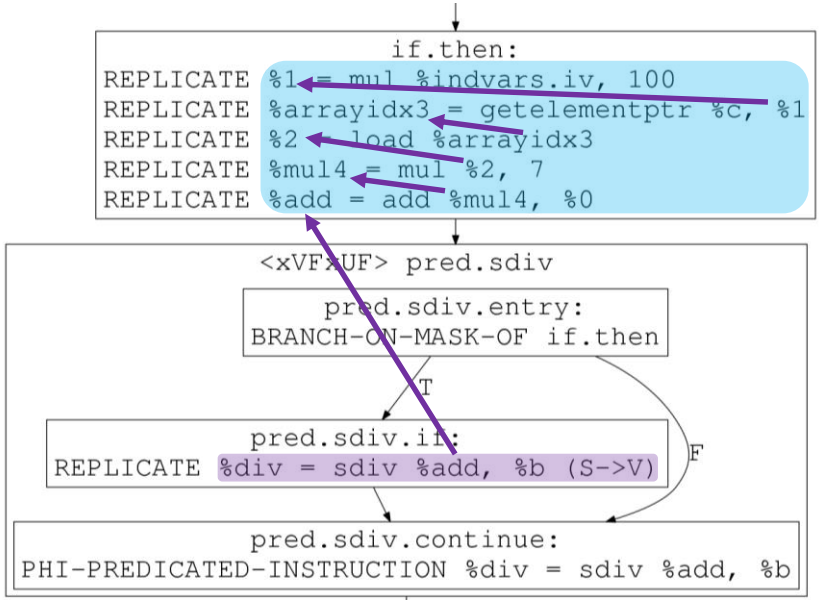
```
// vectorize here
for (i = ilb; i < iub; ++i) {
  …
  for (j = jlb(i); j < jub(i); ++j) {
    while (cond(i, j)) { … }
```

```
julb = hmin(jlb(i));
juub = hmax(jub(i));
cont1 = T;
for (j = julb; j < juub; ++j) {
  if (jlb(i) <= j && j < jub(i) && cont1) {
    cont2 = cond(i, j);
    while (hor(cont2)) {
      if (cont2) {
        …
        cont2 = cond(i, j);
      }
    }
    if (…) cont1 = F;
    if (!hor(cont1)) break;
```

Take Predication Decisions by Transforming One VPlan to Another

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Taking Decision (4/4): SinkScalarOperands



Requires Fine-grain Modeling of Def/Use at instruction-level

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

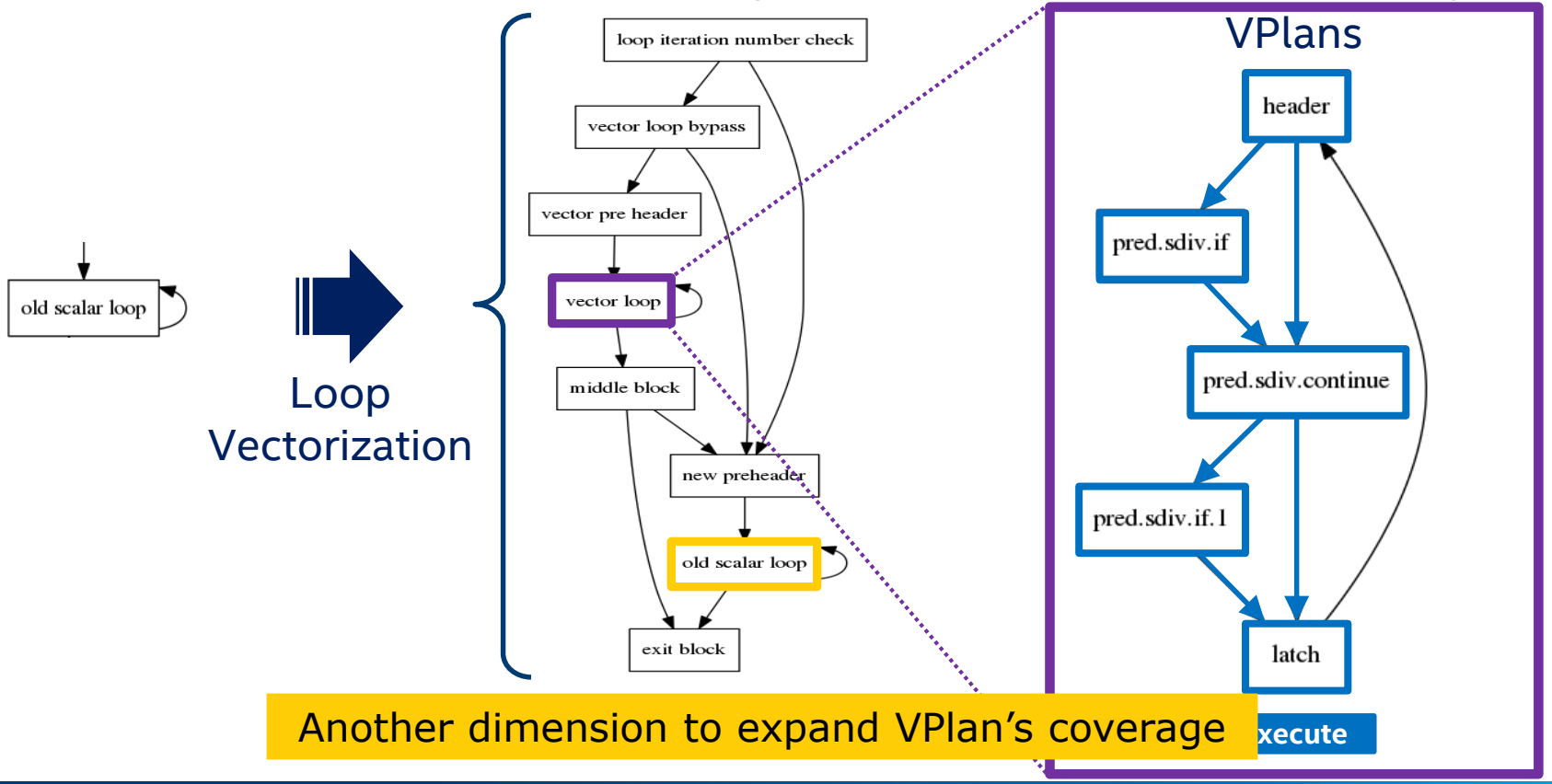
B.3. FROM CARRYING OUT DECISIONS TO MAKING THEM

Use VPlan to also Make Vectorization Decisions

- Instead of first making the decisions, and then using VPlan to carry them out
- Run cost-based analyses on VPlan
  - Based on cost estimates computed by VPlan
  - Based on VPInstruction model
- Apply desired decisions by transforming VPlan, potentially versioning it
  - Based on “what-if” versioning support

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Expand VPlan's Scope Beyond Vector Loop Body



Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Taking Decision (1/4): Interleave Groups – revisit

```
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

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

VPlan for VF=4

```
VPInterleaveRecipe:
  %1 = load %arrayidx3
  %0 = load %arrayidx1

VPWidenRecipe:
  %mul1 = mul %0, 3
  %add4 = add %mul1, %1
  store %add4, %arrayidx5
```

Ingredients

VPlan Execution

IR After Vectorizing for VF=4

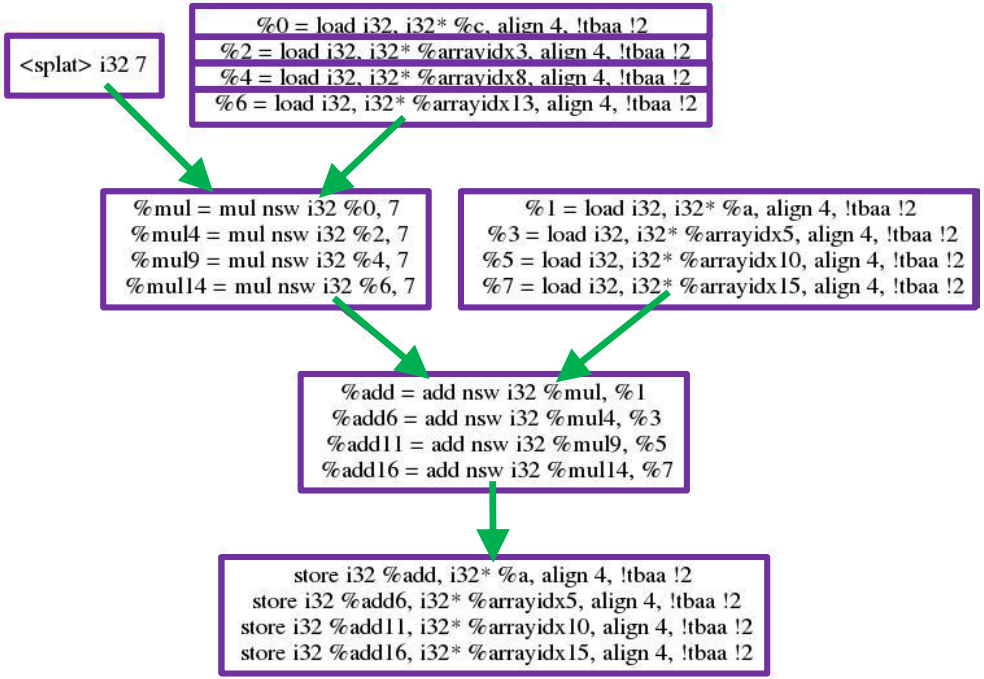
```
vector.body:
  …
  %all = load <8 x i32>, %5
  %even = shufflevector %all, <0,2,4,6>
  %odd = shufflevector %all, <1,3,5,7>
  %6 = mul %odd, <3,3,3,3>
  %9 = add %6, %even
  store %9, %12
  …
```

Combining two `load %0, %1` into one `load %all` – looks familiar?

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

A Model for Vectorized Instructions?

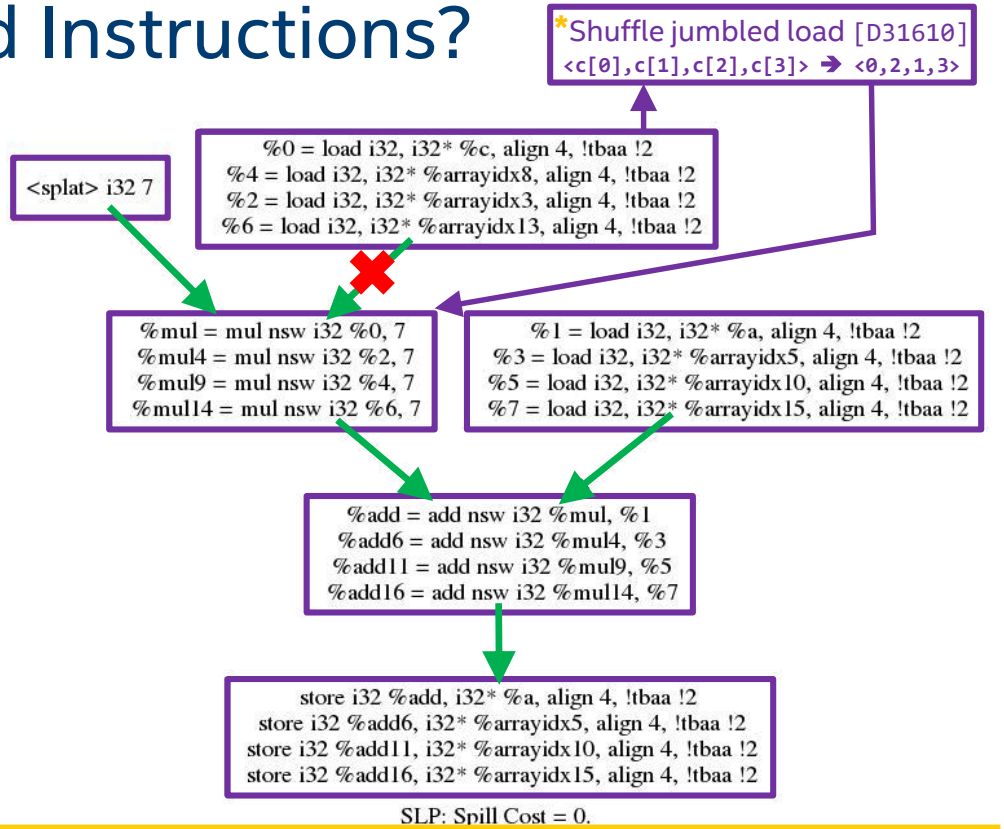
```
void foo(int * restrict a, int b, int *c) {
  a[0] = c[0] * 7 + a[0];
  a[1] = c[2] * 7 + a[1];
  a[2] = c[1] * 7 + a[2];
  a[3] = c[3] * 7 + a[3];
}
```



Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

A Model for Vectorized Instructions?

```
void foo(int * restrict a, int b, int *c) {
  a[0] = c[0] * 7 + a[0];
  a[1] = c[2] * 7 + a[1];
  a[2] = c[1] * 7 + a[2];
  a[3] = c[3] * 7 + a[3];
}
```



Def/Use Model for New & Ingredient-based Instructions & Dependences

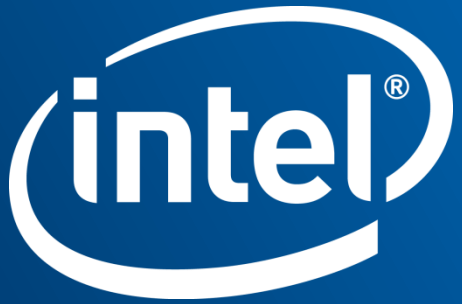
Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Key Takeaways

A. Current State: 1st step introducing VPlan to Loop Vectorizer – committed
   1. Records vectorization decisions in VPlan
   2. Drives vector code generation by executing a VPlan

B. Going Forward: shift vectorization process to be VPlan-based
   1. Refine the model, include masking and break Recipes into VPInstructions
   2. Carry out decisions based on VPlan, in addition to recording them
   3. Make decisions based on VPlan, including legal and cost-based analyses

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.


intel®