SPACEMIT
进迭时空

Introduce the Implementation of LLVM Loop Vectorizer

wengliqin, Compilers@SpacemiT

zhuangqiubin, Compilers@SpacemiT

Loop Vectorizer

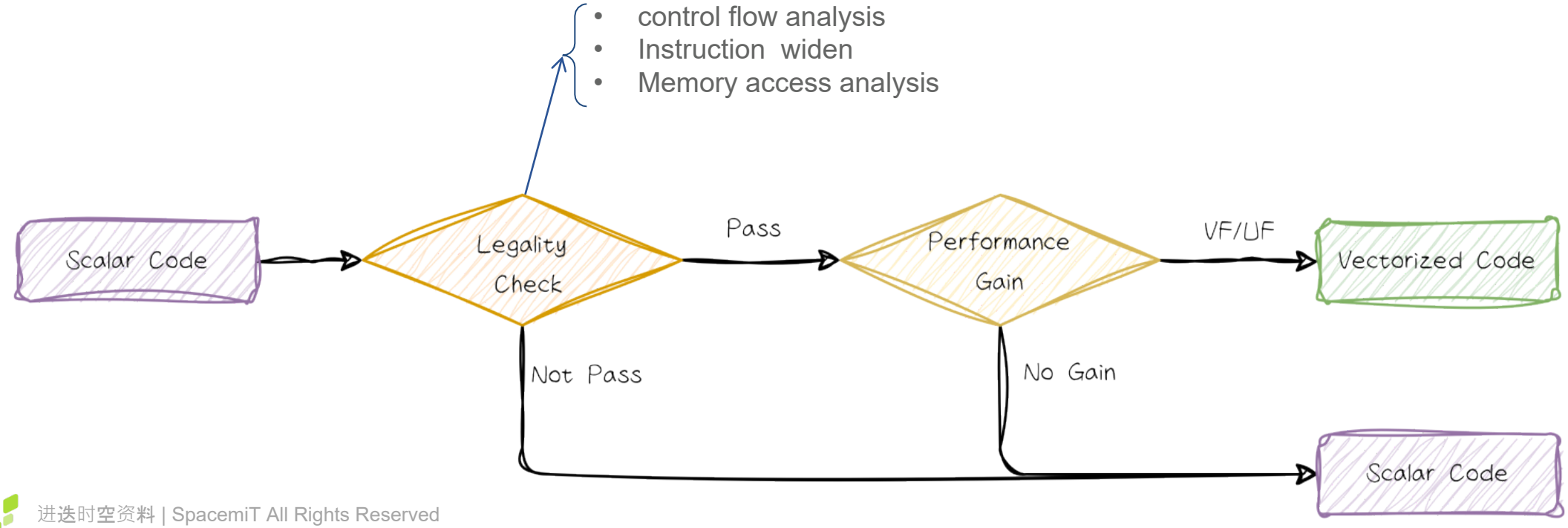
The Loop Vectorizer widens instructions in loops to operate on multiple consecutive iterations

Before Loop Vectorization	After Loop Vectorization

`void bar(int *A, int *B, int *C) {`
`  for (int i = 0; i < 1024; i++)`
`  A[i] = B[i] + C[i];`
`}`
	`void bar(int *A, int *B, int *C) {`
	`  for (int i = 0; i < 256; i += 4) {`
	`  A[i] = B[i] + C[i];`
	`  A[i + 1] = B[i + 1] + C[i + 1];`
	`  A[i + 2] = B[i + 2] + C[i + 2];`
	`  A[i + 3] = B[i + 3] + C[i + 3];`
	`  }`
	`}`

Loop Vectorizer

- Check if the current loop can be vectorized and verify its legality
- If the legality check passes, use the cost model to analyze whether vectorization is beneficial.
- If the vectorized code is profitable, complete the conversion from scalar to vector code and update the curent loop’s control flow.



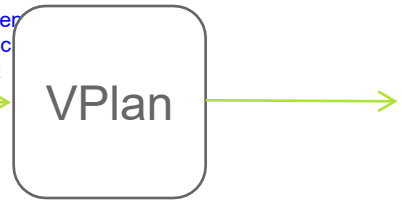
Vectorization Plan in the Loop Vectorizer

VPlan is an explicit model for describing vectorization candidates. It serves two main purposes: optimizing candidates by reliably estimating their cost, and performing their final translation into LLVM IR.

- Simplfied the analysis and transforms
- Vectorize loops better, and vectorize more loops
- More accurate in cost model for RecipeCost

Input LLVM-IR

```
entry:....

for.body:  %iv = phi i64 [ 0, %for.body.preheader ], [ %iv.next, %for.body ]  %gep = getelemer
nuw i32, ptr %A, i64 %iv  %0 = load i32, ptr %gep  %inc = add nsw i32 %0, 1  store i32 %inc
%iv.next = add nuw nsw i64 %iv, 1  %exitcond.not = icmp eq i64 %iv.next, %wide.trip.count

 br i1 %exitcond.not, label %for.cond.cleanup.loopexit, label %for.body

 ....
```



Output LLVM-IR

```
vector.phy:
....

vector.body:  %index = phi i64 [ 0, %vector.ph ], [ %index.next, %vector.body ]  %evl.based.iv = phi i64 [ 0,
%vector.ph ], [ %index.evl.next, %vector.body ]  %avl = sub i64 %wide.trip.count, %evl.based.iv  %5 = call i32
@llvm.experimental.get.vector.length.i64(i64 %avl, i32 4, i1 true)  %6 = getelementptr inbounds nuw i32, ptr %A,
i64 %evl.based.iv  %7 = getelementptr inbounds nuw i32, ptr %6, i32 0  %vp.op.load = call <vscale x 4 x i32>
@llvm.vp.load.nxv4i32.p0(ptr %7, <vscale x 4 x i1> splat (i1 true), i32 %5)  %8 = add nsw <vscale x 4 x i32>
%vp.op.load, splat (i32 1)  call void @llvm.vp.store.nxv4i32.p0(<vscale x 4 x i32> %8, ptr %7, <vscale x 4 x i1>
splat (i1 true), i32 %5)  br i1 %10, label %middle.block, label %vector.body


.middle.block  ....
```

Vectorization Plan in the Loop Vectorizer

Input LLVM-IR

```
entry:  %cmp4 = icmp sgt i32 %wide.trip.count, 0  br i1 %cmp4, label %for.body.preheader, label
%for.cond.cleanup for.body.preheader:  br label %for.body for.body:  %iv = phi i64 [ 0, %for.body.preheader
], [ %iv.next, %for.body ]  %gep = getelementptr inbounds nuw i32, ptr %A, i64 %iv  %0 = load i32, ptr %gep
%inc = add nsw i32 %0, 1  store i32 %inc, ptr %gep  %iv.next = add nuw nsw i64 %iv, 1  %exitcond.not =
icmp eq i64 %iv.next, %wide.trip.count

  br i1 %exitcond.not, label %exit, label %for.body

exit:  ret void
```

→ Ingredient

VPlan

```
VPlan 'Initial VPlan for VF={vscale x 1,vscale x 2,vscale x 4},UF={1}' {Live-in vp<%0> = VFLive-in vp<%1> =
VF * UFLive-in vp<%2> = vector-trip-countvp<%4> = original trip-count

ir-bb<for.body.preheader>:Successor(s): scalar.ph, vector.ph  vector.ph:Successor(s): vector loop  <x1>
vector loop: {  vector.body:    EMIT vp<%5> = CANONICAL-INDUCTION ir<0>, vp<%index.next>
EXPLICIT-VECTOR-LENGTH-BASED-IV-PHI vp<%6> = phi ir<0>, vp<%index.evl.next>    EMIT vp<%avl>
= sub vp<%4>, vp<%6>    EMIT vp<%7> = EXPLICIT-VECTOR-LENGTH vp<%avl>    vp<%8> = SCALAR-
STEPS vp<%6>, ir<1>, vp<%0>    CLONE ir<%gep> = getelementptr inbounds nuw ir<%A>, vp<%8>
vp<%9> = vector-pointer ir<%gep>    WIDEN ir<%0> = vp.load vp<%9>, vp<%7>    WIDEN ir<%inc> = add
nsw ir<%0>, ir<1>    vp<%10> = vector-pointer ir<%gep>    WIDEN vp.store vp<%10>, ir<%inc>, vp<%7>
EMIT vp<%11> = zext vp<%7> to i64    EMIT vp<%index.evl.next> = add nuw vp<%11>, vp<%6>    EMIT
vp<%index.next> = add nuw vp<%5>, vp<%1>    EMIT branch-on-count vp<%index.next>, vp<%2>  No
successors}Successor(s): middle.block middle.block...

scalar.ph:  EMIT-SCALAR vp<%bc.resume.val> = phi [ ir<0>, ir-bb<for.body.preheader> ]Successor(s): ir-
bb<for.body>  ir-bb<for.body>:...}
```

VPRBasicBlocks

VPBasicBlocks

→ Recipe

VPRegionBlock

VPlan model of the input IR

Plain CFG is constructed from the Input LLVM-IR CFG

- Create a VPBB for each BB and link it to its successor and predecessor VPBBs
- Create a corresponding VPInstruction for BB's Instruction

```
entry:  %cmp4 = icmp sgt i32 %wide.trip.count, 0  br i1 %cmp4, label %for.body.preheader, label
%for.cond.cleanup for.body.preheader:  br label %for.body  for.body:  %iv = phi i64 [ 0, %for.body.preheader
], [ %iv.next, %for.body ]  %gep = getelementptr inbounds nuw i32, ptr %A, i64 %iv  %0 = load i32, ptr %gep
%inc = add nsw i32 %0, 1  store i32 %inc, ptr %gep  %iv.next = add nuw nsw i64 %iv, 1  %exitcond.not =
icmp eq i64 %iv.next, %wide.trip.count

  br i1 %exitcond.not, label %exit, label %for.body

exit:  ret void
```

Transform

```
VPlan 'Plain CFG for UF>=1' {

ir-bb<for.body.preheader>:  Successor(s): for.body  for.body:  WIDEN-PHI ir<%iv> = phi [ ir<0>, ir-
bb<for.body.preheader> ], [ ir<%iv.next>, for.body ]

  EMIT ir<%gep> = getelementptr ir<%A>, ir<%iv>  EMIT ir<%0> = load ir<%gep>  EMIT ir<%inc> = add ir<%0>,
ir<1>  EMIT store ir<%inc>, ir<%gep>  EMIT ir<%iv.next> = add ir<%iv>, ir<1>  EMIT ir<%exitcond.not> = icmp
ir<%iv.next>, ir<%wide.trip.count>  EMIT branch-on-cond ir<%exitcond.not>

Successor(s): ir-bb<exit>, for.body  ir-bb<exit>:No successors}
```

VPlan to VPlan Transforms: Prepare for Vectorization

- Canonical the header and latch of loop
- Insert necessary Plain CFG VPBB, such as vector.ph/middle.block/scalar.ph
- Set vector trip count, and calculate the step for each vector iteration

```
VPlan 'Plain CFG for UF>=1' { ir-bb<for.body.preheader>:Successor(s): for.body for.body:  WIDEN-PHI ir<%iv> =
phi [ ir<0>, ir-bb<for.body.preheader> ], [ ir<%iv.next>, for.body ]

  EMIT ir<%gep> = getelementptr ir<%A>, ir<%iv>  EMIT ir<%0> = load ir<%gep>  EMIT ir<%inc> = add ir<%0>,
ir<1>  EMIT store ir<%inc>, ir<%gep>  EMIT ir<%iv.next> = add ir<%iv>, ir<1>  EMIT ir<%exitcond.not> = icmp
ir<%iv.next>, ir<%wide.trip.count>  EMIT branch-on-cond ir<%exitcond.not>Successor(s): ir-
bb<for.cond.cleanup.loopexit>, for.body ir-bb<for.cond.cleanup.loopexit>:No successors}
```

Transform

```
VPlan ' for UF>=1' {
Live-in vp<%0> = VF * UF
Live-in vp<%1> = vector-trip-countvp<%2> = original trip-count.....Successor(s): scalar.ph, vector.ph


vector.ph:Successor(s): for.body for.body:  EMIT vp<%3> = CANONICAL-INDUCTION ir<0>,
vp<%index.next>

.....  EMIT vp<%index.next> = add nuw vp<%3>, vp<%0>

  EMIT branch-on-count vp<%index.next>, vp<%1>
Successor(s): middle.block, for.body

middle.block:  EMIT branch-on-cond ir<true>.....

scalar.ph:Successor(s): ir-bb<for.body>ir-bb<for.body>:.....No successors}
```

VPlan to VPlan Transforms: Create Vector Region

Create Vector Region for vector body

VPlan ' for UF>=1' {.... vector.ph:Successor(s): for.body

for.body: EMIT vp<%3> = CANONICAL-INDUCTION ir<0>, vp<%iv.next> WIDEN-PHI ir<%iv> = phi [ir<0>, vector.ph], [ir<%iv.next>, for.body] .... EMIT vp<%iv.next> = add nuw vp<%3>, vp<%0> EMIT branch-on-count vp<%iv.next>, vp<%1>Successor(s): middle.block, for.body

Transform

vector.ph:Successor(s): vector loop

<x1> vector loop: { vector.body: EMIT vp<%3> = CANONICAL-INDUCTION ir<0>, vp<%iv.next> .... EMIT vp<%i.next> = add nuw vp<%3>, vp<%0> EMIT branch-on-count vp<%iv.next>, vp<%1> No successors

}Successor(s): middle.block

VPlan to VPlan Transforms: Introduce Masks And Linearize

- Mask generations: Generates masks for each block and edge, supporting predicated control flow
- CFG linearization: Flattens complex control flows into a single linked list for easy vectorization, such as if-else, switch..

```
VPlan ' for UF>=1' {....

<x1> vector loop: {  vector.body:    EMIT vp<%3> = CANONICAL-INDUCTION ir<0>, vp<%index.next>
WIDEN-PHI ir<%iv> = phi [ ir<0>, vector.ph ], [ ir<%iv.next>, vector.body ]    ....    EMIT vp<%index.next> =
add nuw vp<%3>, vp<%0>    EMIT branch-on-count vp<%index.next>, vp<%1>  No successors}...}
```

Transform

```
VPlan ' for UF>=1' {....

<x1> vector loop: {  vector.body:    EMIT vp<%4> = CANONICAL-INDUCTION ir<0>, vp<%index.next>
WIDEN-PHI ir<%indvars.iv> = phi [ ir<0>, vector.ph ], [ ir<%iv.next>, vector.body ]

   EMIT vp<%5> = WIDEN-CANONICAL-INDUCTION vp<%4>    EMIT vp<%6> = icmp ule vp<%5>,
vp<%2>

    ....EMIT vp<%index.next> = add nuw vp<%4>, vp<%0>    EMIT branch-on-count vp<%index.next>,
vp<%1>  No successors}....}
```

VPlan to VPlan Transforms: Widen Recipe

Choose different widening strategies for various types of instructions, such as phi/load/store/call....

Uniform/Scalar/Vectorize in the vectorized loop

```
VPlan ' for UF>=1' {
Live-in vp<%0> = VF * UF
Live-in vp<%1> = vector-trip-count
Live-in vp<%2> = backedge-taken count
vp<%3> = original trip-count
 ....

<x1> vector loop: {  vector.body:    EMIT vp<%4> = CANONICAL-INDUCTION ir<0>, vp<%index.next>
WIDEN-PHI ir<%iv> = phi [ ir<0>, vector.ph ], [ ir<%index.next>, vector.body ]    EMIT vp<%5> = WIDEN-
CANONICAL-INDUCTION vp<%4>    EMIT vp<%6> = icmp ule vp<%5>, vp<%2>    EMIT ir<%gep> =
getelementptr ir<%A>, ir<%iv>    EMIT ir<%0> = load ir<%gep>    EMIT ir<%inc> = add ir<%0>, ir<1>
EMIT store ir<%inc>, ir<%gep>    EMIT ir<%iv.next> = add ir<%iv>, ir<1>    EMIT ir<%exitcond.not> = icmp
ir<%iv.next>, ir<%wide.trip.count>    EMIT vp<%index.next> = add nuw vp<%4>, vp<%0>    EMIT branch-on
-count vp<%index.next>, vp<%1>  No successors} ....}
```

Transform

```
VPlan 'Initial VPlan for VF={vscale x 1,vscale x 2,vscale x 4},UF>=1' {Live-in vp<%0> = VF

Live-in vp<%1> = VF * UF
Live-in vp<%2> = vector-trip-count
Live-in vp<%3> = backedge-taken count
vp<%4> = original trip-count

....
<x1> vector loop: {  vector.body:    EMIT vp<%5> = CANONICAL-INDUCTION ir<0>, vp<%index.next>

   ir<%iv> = WIDEN-INDUCTION  ir<0>, ir<1>, vp<%0>    EMIT vp<%6> = WIDEN-CANONICAL-
INDUCTION vp<%5>    EMIT vp<%7> = icmp ule vp<%6>, vp<%3>

   CLONE ir<%gep> = getelementptr inbounds nuw ir<%A>, ir<%iv>    vp<%8> = vector-pointer ir<%gep>
WIDEN ir<%0> = load vp<%8>, vp<%7>    WIDEN ir<%inc> = add nsw ir<%0>, ir<1>

   vp<%9> = vector-pointer ir<%gep>    WIDEN store vp<%9>, ir<%inc>, vp<%7>    CLONE ir<%iv.next> =
add nuw nsw ir<%iv>, ir<1>    CLONE ir<%exitcond.not> = icmp eq ir<%iv.next>, ir<%wide.trip.count>

   EMIT vp<%index.next> = add nuw vp<%5>, vp<%1>    EMIT branch-on-count vp<%index.next>, vp<%2>
No successors}....}
```

VPlan to VPlan Transforms: Recipe to EVLRecipe

Tail Folding style is EVL

- Insert ExplicitVectorLength of VPIntrinsic which controls the number of elements processed per loop iteration
- LLVM intrinsics Recipe transform to vp intrinsics Recipe, only support ld/st/select/reduction

```
VPlan 'Initial VPlan for VF={vscale x 1,vscale x 2,vscale x 4},UF>=1' {....<x1> vector loop: {  vector.body:
EMIT vp<%5> = CANONICAL-INDUCTION ir<0>, vp<%iv.next>    ir<%iv> = WIDEN-INDUCTION  ir<0>,
ir<1>, vp<%0>    EMIT vp<%6> = WIDEN-CANONICAL-INDUCTION vp<%5>    EMIT vp<%7> = icmp ule
vp<%6>, vp<%3>    CLONE ir<%gep> = getelementptr inbounds nuw ir<%A>, ir<%iv>    vp<%8> = vector-
pointer ir<%gep>    WIDEN ir<%0> = load vp<%8>, vp<%7>    WIDEN ir<%inc> = add nsw ir<%0>, ir<1>
vp<%9> = vector-pointer ir<%gep>    WIDEN store vp<%9>, ir<%inc>, vp<%7>    CLONE ir<%iv.next> =
add nuw nsw ir<%iv>, ir<1>    CLONE ir<%exitcond.not> = icmp eq ir<%iv.next>, ir<%wide.trip.count>
EMIT vp<%iv.next> = add nuw vp<%5>, vp<%1>    EMIT branch-on-count vp<%iv.next>, vp<%2>  No
successors}....}
```

Transform

```
VPlan 'Initial VPlan for VF={vscale x 1,vscale x 2,vscale x 4},UF={1}' {....<x1> vector loop: {  vector.body:
EMIT vp<%5> = CANONICAL-INDUCTION ir<0>, vp<%iv.next>    EXPLICIT-VECTOR-LENGTH-BASED-IV-
PHI vp<%6> = phi ir<0>, vp<%evl.next>

    EMIT vp<%avl> = sub vp<%4>, vp<%6>    EMIT vp<%7> = EXPLICIT-VECTOR-LENGTH vp<%avl>
vp<%8> = SCALAR-STEPS vp<%6>, ir<1>, vp<%0>    CLONE ir<%gep> = getelementptr inbounds nuw
ir<%A>, vp<%8>    vp<%9> = vector-pointer ir<%gep>    WIDEN ir<%0> = vp.load vp<%9>, vp<%7>

    WIDEN ir<%inc> = add nsw ir<%0>, ir<1>    vp<%10> = vector-pointer ir<%gep>

    WIDEN vp.store vp<%10>, ir<%inc>, vp<%7>    EMIT vp<%11> = zext vp<%7> to i64    EMIT
vp<%evl.next> = add nuw vp<%11>, vp<%6>    EMIT vp<%iv.next> = add nuw vp<%5>, vp<%1>    EMIT
branch-on-count vp<%iv.next>, vp<%2>  No successors}....}
```

VPlan model of the output vector IR

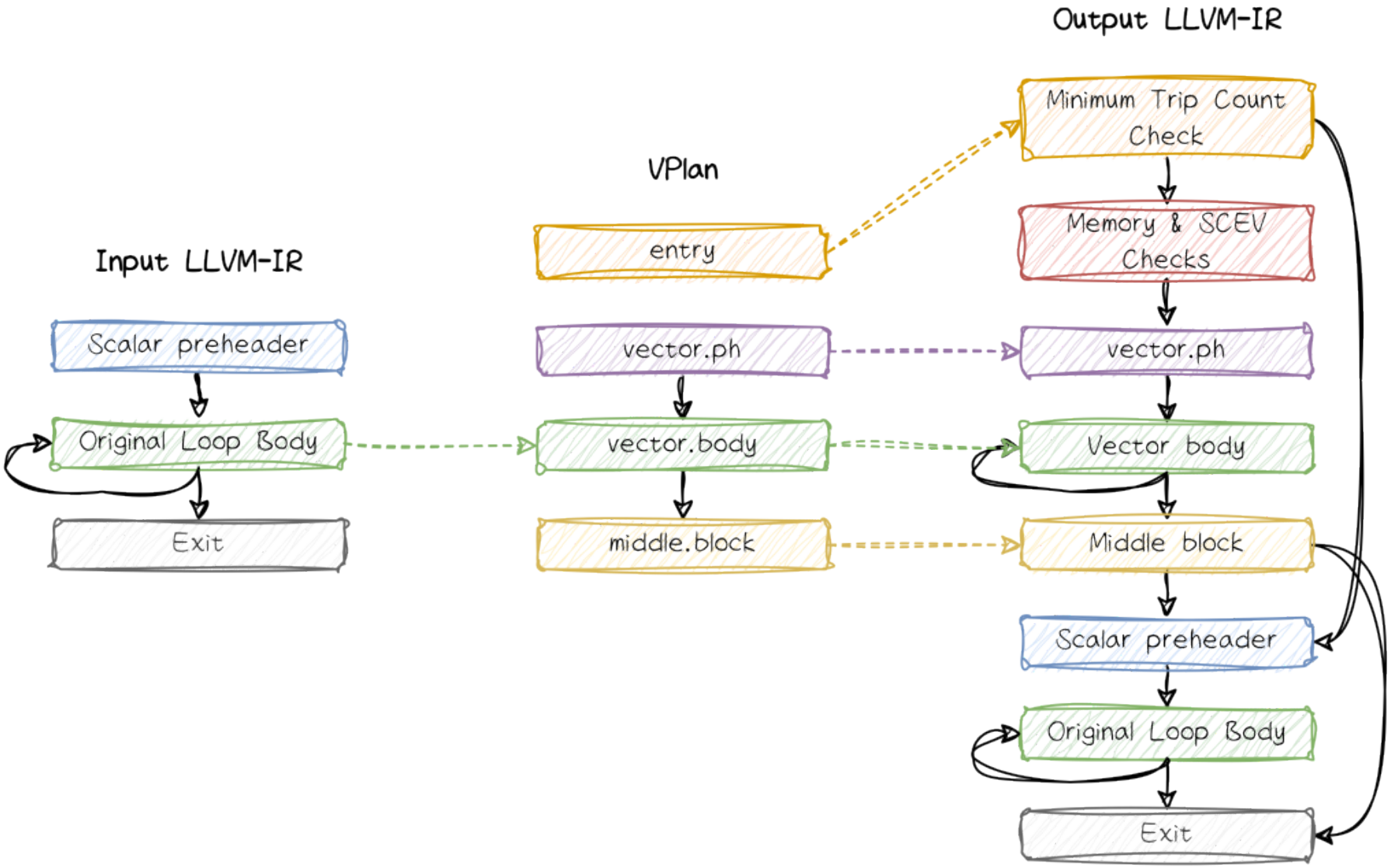
Get the best VF through the instruction cost information of the corresponding backend platform. If VF > 1, convert it into vectorized code

```
VPlan 'Initial VPlan for VF={vscale x 1,vscale x 2,vscale x 4},UF={1}' {....vector.ph:Successor(s): vector loop
<x1> vector loop: {  vector.body:    EMIT vp<%5> = CANONICAL-INDUCTION ir<0>, vp<%index.next>
EXPLICIT-VECTOR-LENGTH-BASED-IV-PHI vp<%6> = phi ir<0>, vp<%index.evl.next>    EMIT vp<%avl>
= sub vp<%4>, vp<%6>    EMIT vp<%7> = EXPLICIT-VECTOR-LENGTH vp<%avl>    vp<%8> = SCALAR-
STEPS vp<%6>, ir<1>, vp<%0>    CLONE ir<%gep> = getelementptr inbounds nuw ir<%A>, vp<%8>
vp<%9> = vector-pointer ir<%gep>    WIDEN ir<%0> = vp.load vp<%9>, vp<%7>    WIDEN ir<%inc> = add
nsw ir<%0>, ir<1>    vp<%10> = vector-pointer ir<%gep>    WIDEN vp.store vp<%10>, ir<%inc>, vp<%7>
EMIT vp<%11> = zext vp<%7> to i64    EMIT vp<%index.evl.next> = add nuw vp<%11>, vp<%6>    EMIT
vp<%index.next> = add nuw vp<%5>, vp<%1>    EMIT branch-on-count vp<%index.next>, vp<%2>  No
successors}....}
```

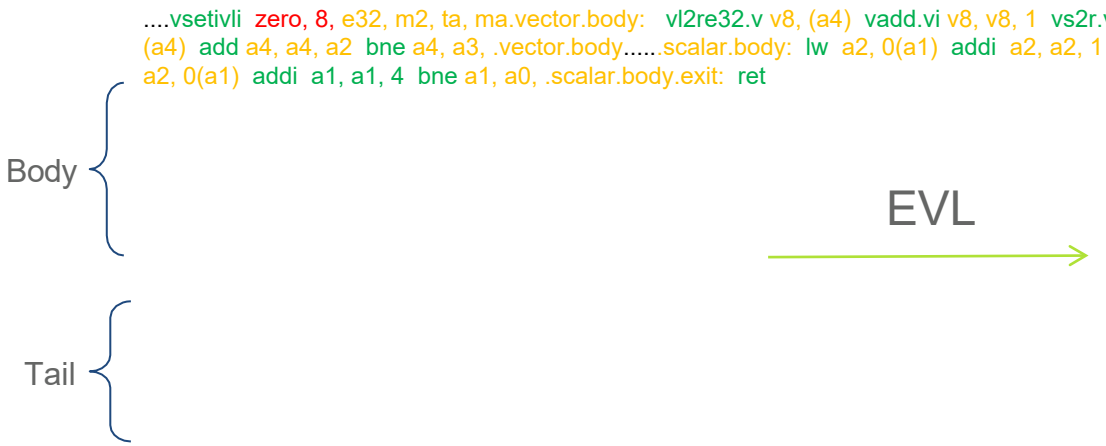
Transform

```
vector.ph:....  br label %vector.body


vector.body:
  %index = phi i64 [ 0, %vector.ph ], [ %index.next, %vector.body ]  %evl.based.iv = phi i64 [ 0, %vector.ph ], [ %index.evl.next, %vector.b
]  %avl = sub i64 %wide.trip.count, %evl.based.iv  %5 = call i32 @llvm.experimental.get.vector.length.i64(i64 %avl, i32 4, i1 true)  %6 =
getelementptr inbounds nuw i32, ptr %A, i64 %evl.based.iv  %7 = getelementptr inbounds nuw i32, ptr %6, i32 0  %vp.op.load = call <vsc
x 4 x i32> @llvm.vp.load.nxv4i32.p0(ptr %7, <vscale x 4 x i1> splat (i1 true), i32 %5)  %8 = add nsw <vscale x 4 x i32> %vp.op.load, spla
(i32 1)  call void @llvm.vp.store.nxv4i32.p0(<vscale x 4 x i32> %8, ptr %7, <vscale x 4 x i1> splat (i1 true), i32 %5)  %9 = zext i32 %5 to i
%index.evl.next = add nuw i64 %9, %evl.based.iv  %index.next = add nuw i64 %index, %4  %10 = icmp eq i64 %index.next, %n.vec

  br i1 %10, label %middle.block, label %vector.body

middle.block:   ....
```

VPlan Status Updata: Refactor Initial VPlan Creation



VPlan Status Updata: Tail Folding style is EVL

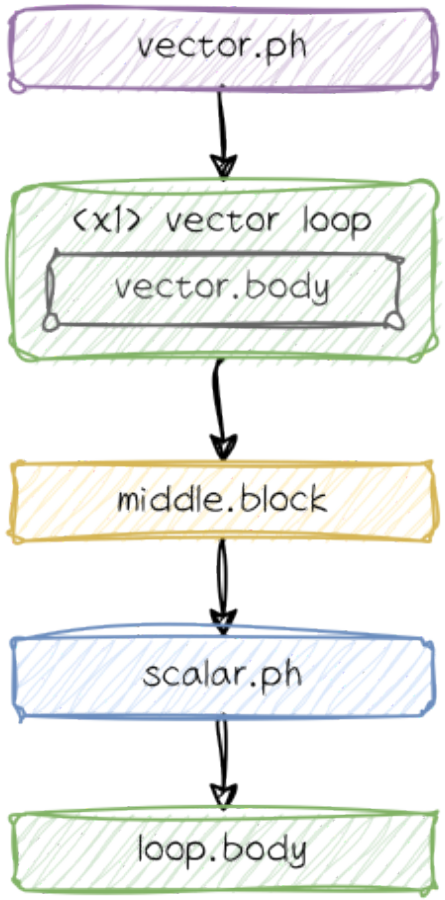


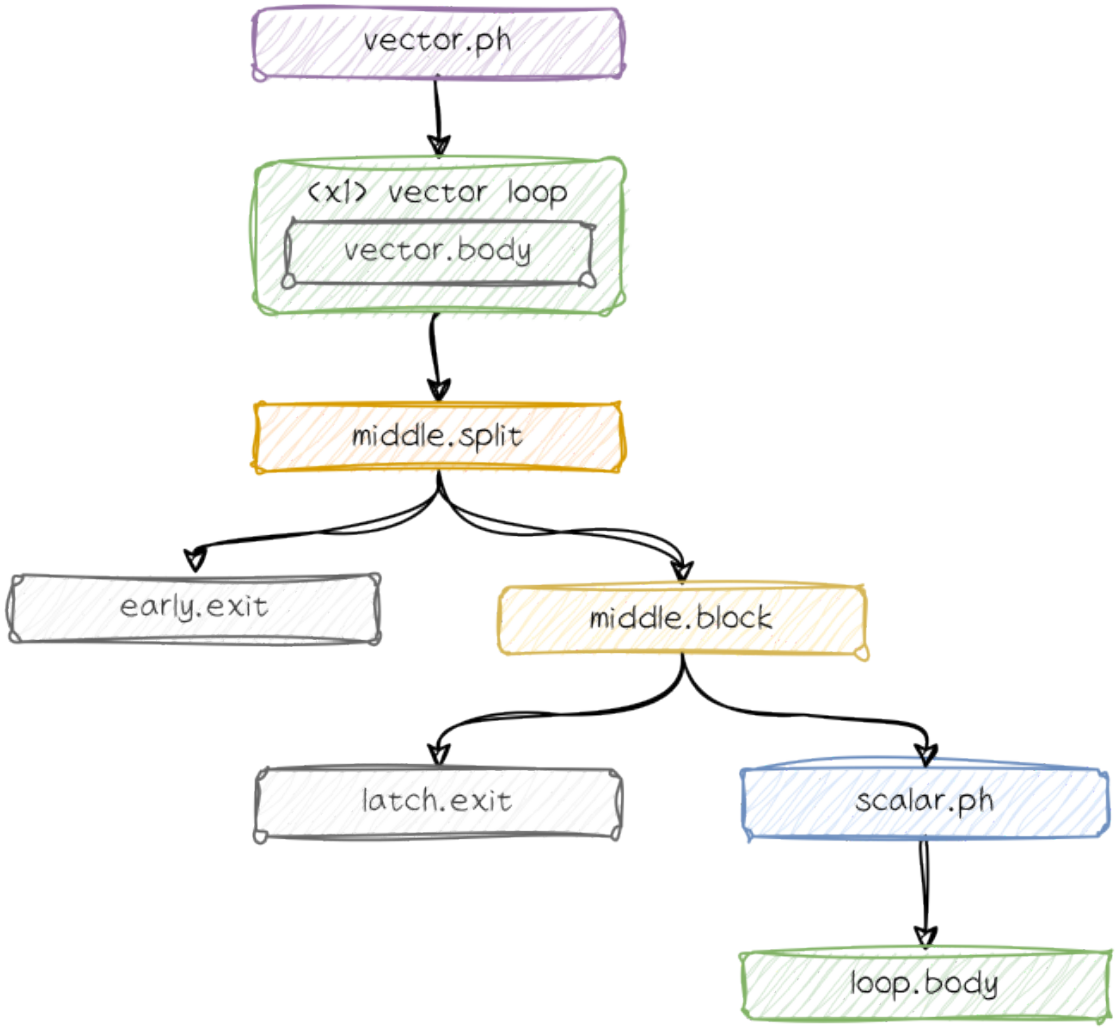
Options:

-mllvm -force-tail-folding-style=data-with-evl

-mllvm -prefer-predicate-over-epilogue=prefer-predicate-over-epilogue

VPlan Status Updata: Vectorizeing multi exit loops





Options:

-mllvm -enable-early-exit-vectorization=true

VPlan RoadMap

- Gradually Improve the cost model based VPlan
- LoopVectorize with EVL enable UF
- Improvements to RISC-V Vector code generation in LLVM
- .....

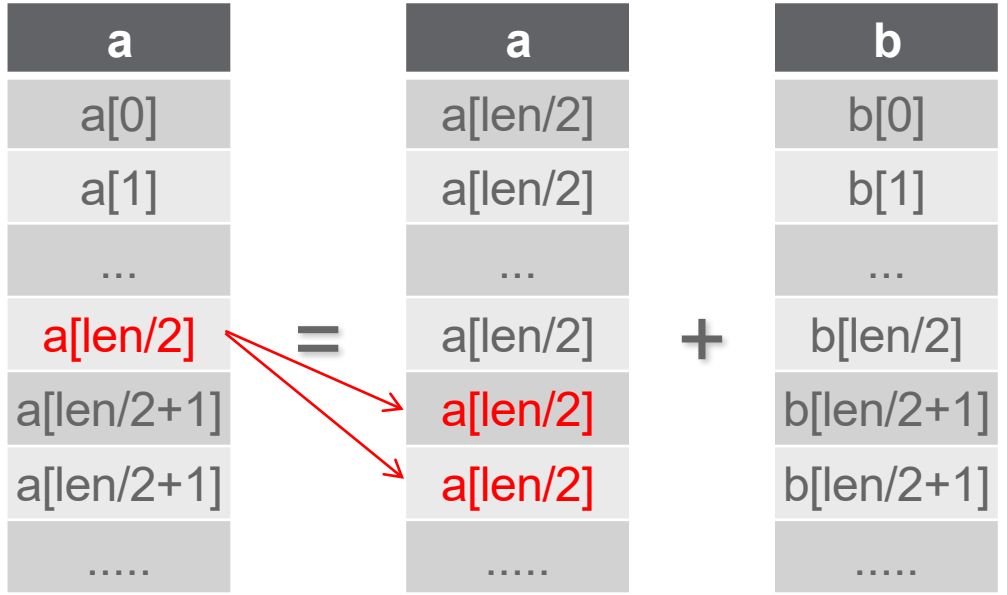
SPEC2017 Perforance

SPEC2017

lowcer is better

Example In TSVC

```
#define iterations 100000#define len 32000float *a;float
*b;void s113() {  for (int n = 0; n < 2 * iterations; n++)    for (int i
= 0; i < len; i++)      a[i] = a[len/2] + b[i];}
```

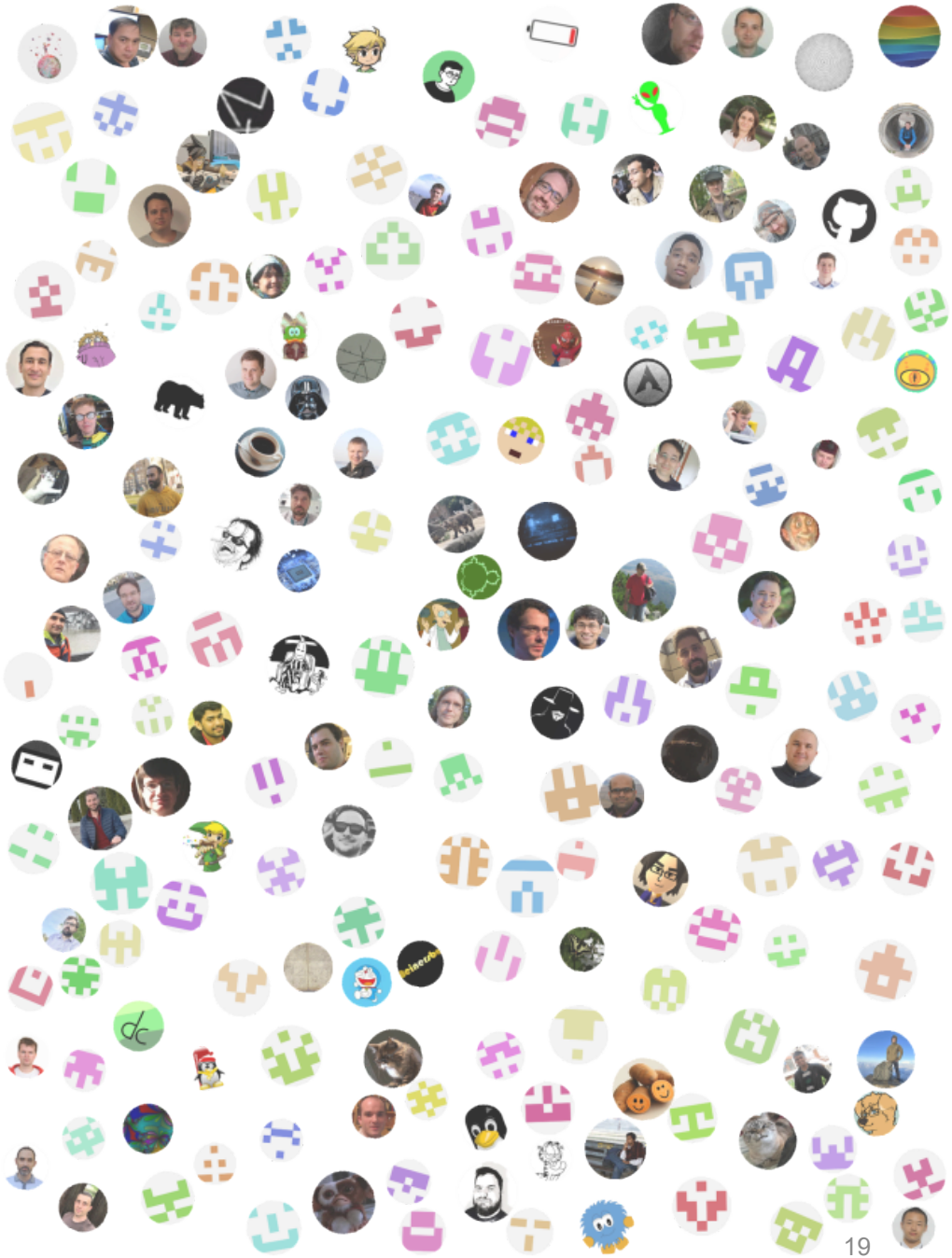


Memory dependence in a[len/2], and prevents vectorization

- Splitting the iteration space
- Extract a[len/2] variables put in outside the loop

Get Involved and Stay Updated

- https://www.llvm.org/docs/VectorizationPlan.html
- https://discourse.llvm.org/t/vplan-progress-update-august-2023/73003
- https://lnt.lukelau.me/db_default/v4/nts/471?compare_to=405
- https://github.com/llvm/llvm-project/pull/137343
- https://github.com/llvm/llvm-project/pull/137343
- https://github.com/llvm/llvm-project/pull/144564
- https://forum.spacemit.comc



SPACEMIT
进迭时空

Thanks

以RISC-V架构数智未来

RISC-V ARCHITECTURE FOR INTELLIGENT FUTURE

进迭时空（杭州）科技有限公司
SPACEMIT (Hangzhou) Technology Co., Ltd

电话：0571-89000775
网址：www.spacemit.com
地址：浙江省杭州市余杭区五常街道未来天空中心b座701