An approach to sustainably add Vector Predication to the Loop Vectorizer

Lorenzo Albano <lorenzo.albano@bsc.es>
Roger Ferrer <roger.ferrer@bsc.es>

Background

The Loop Vectorizer does not make use of Vector Predication intrinsics. We have been extending the Loop Vectorizer with new Vector Planner recipes that are vector length and mask aware. This approach works but leads to recipes duplication in the Loop Vectorizer.

We wanted to see if a simpler approach is workable.

Method

We implemented a new mode of tail folding in the Loop Vectorizer to only emit the minimum Vector Predication intrinsics needed for correctness: memory accesses and a (target-dependent) set vector length mechanism that depends on the remaining iterations of the loop.

A later pass analyses what is demanded from the vectors, starting from Vector Predication stores. From this analysis, vectorized IR instructions (when possible) are replaced with Vector Predication intrinsics that use the demanded vector length and mask.

Results

We have used the TSVC-2 benchmark and the RISC-V target. We have compared the emitted code against our earlier, more invasive, implementation. The emitted code is comparable to our previous implementation.

Conclusion

This approach requires a small set of changes to the Loop Vectorizer while allowing us to reason about vector length and predicate in another pass.

This approach is low cost and benefits RISC-V and VE targets, that have vector length, and SVE and AVX-512 targets that can now make a more effective use of their predicated ISAs.

Our previous approach adds many Vector Planner Recipes

+ VPPredicatedBlendRecipe
+ VPPredicatedFirstOrderRecurrencePHIRecipe
+ VPPredicatedReductionRecipe
+ VPPredicatedWidenCallRecipe
+ VPPredicatedWidenMemoryInstructionRecipe
+ VPPredicatedWidenMemoryRecipe
+ VPPredicatedWidenRecipe
+ VPPredicatedWidenSelectRecipe

VPInstructions:

+ PredicatedFirstOrderRecurrenceSplice
+ VPICmpULE
+ VPNot
+ VPOr
+ VPSelect

Previous Approach

```
vector.ph:
Successor(s): vector loop
<x1> vector loop
vector.body:
  EMIT vp<%3> = CANONICAL-INDUCTION
  EMIT vp<%4> = GENERATE-EXPLICIT-VECTOR-LENGTH vp<%2>
  vp<%5>  = SCALAR-STEPS vp<%3>, ir<1>
  EMIT vp<%6> = WIDEN-CANONICAL-INDUCTION vp<%3>, vp<%4>
  EMIT vp<%7> = vp icmp ule vp<%6> vp<%1> vp<%4>
  CLONE ir<%arrayidx> = getelementptr ir<%a>, vp<%5>
  PREDICATED-WIDEN ir<%0> = load ir<%arrayidx>, vp<%7>, vp<%4> (ALL-ONES-MASK)
  CLONE ir<%arrayidx1> = getelementptr ir<%b>, vp<%5>
  PREDICATED-WIDEN ir<%1> = load ir<%arrayidx1>, vp<%7>, vp<%4> (ALL-ONES-MASK)
  PREDICATED-WIDEN ir<%add> = fadd ir<%0>, ir<%1>, vp<%7>, vp<%4>
  CLONE ir<%arrayidx2> = getelementptr ir<%c>, vp<%5>
  PREDICATED-WIDEN store ir<%arrayidx2>, ir<%add>, vp<%7>, vp<%4> (ALL-ONES-MASK)
  EMIT vp<%14> = VF * UF +  vp<%3> vp<%4>
  EMIT branch-on-count  vp<%14> vp<%0>
No successors
middle.block:
No successors
```

Our current approach uses fewer changes

↻ VPWidentIntOrFPInductionRecipe
↻ VPWidenMemoryInstructionRecipe
+ VPEVLPHIRecipe

VPInstructions:

+ NextEVL

Current Approach

```
vector.ph:
Successor(s): vector loop
<x1> vector loop
vector.body:
  EMIT vp<%2> = CANONICAL-INDUCTION
  EVL-PHI vp<%3> = phi vp<%1>, vp<%14>
  vp<%4>  = SCALAR-STEPS vp<%2>, ir<1>
  EMIT vp<%5> = WIDEN-CANONICAL-INDUCTION vp<%2>
  EMIT vp<%6> = active lane mask vp<%5> vp<%1>
  CLONE ir<%arrayidx> = getelementptr ir<%a>, vp<%4>
  WIDEN ir<%0> = load ir<%arrayidx>, vp<%6>, vp<%3>
  CLONE ir<%arrayidx1> = getelementptr ir<%b>, vp<%4>
  WIDEN ir<%1> = load ir<%arrayidx1>, vp<%6>, vp<%3>
  WIDEN ir<%add> = fadd ir<%0>, ir<%1>
  CLONE ir<%arrayidx2> = getelementptr ir<%c>, vp<%4>
  WIDEN store ir<%arrayidx2>, ir<%add>, vp<%6>, vp<%3>
  EMIT vp<%13> = VF * UF +  vp<%2> vp<%3>
  EMIT vp<%14> = next-evl vp<%3> vp<%13>
  EMIT branch-on-count  vp<%13> vp<%0>
No successors
middle.block:
No successors
```

Upstream tail folding (for comparison)

```
void add_ref(long N, double *c, double *a, double *b) {
  long i;
  for (i = 0; i < N; i++)
    c[i] = a[i] + b[i];
}
```

Upstream (Tail folding)

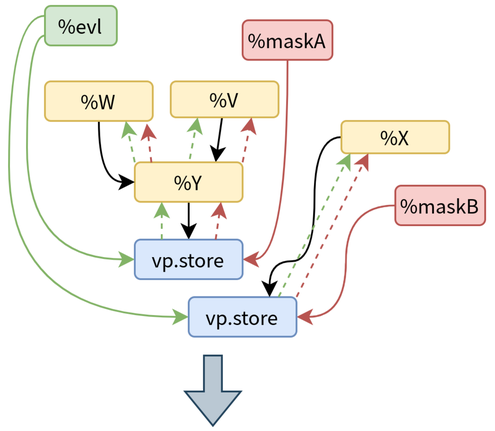
```
vector.ph:
Successor(s): vector loop
<x1> vector loop
vector.body:
  EMIT vp<%2> = CANONICAL-INDUCTION
  vp<%3>  = SCALAR-STEPS vp<%2>, ir<1>
  EMIT vp<%4> = active lane mask vp<%3> vp<%1>
  CLONE ir<%arrayidx> = getelementptr ir<%a>, vp<%3>
  WIDEN ir<%0> = load ir<%arrayidx>, vp<%4>
  CLONE ir<%arrayidx1> = getelementptr ir<%b>, vp<%3>
  WIDEN ir<%1> = load ir<%arrayidx1>, vp<%4>
  WIDEN ir<%add> = fadd ir<%0>, ir<%1>
  CLONE ir<%arrayidx2> = getelementptr ir<%c>, vp<%3>
  WIDEN store ir<%arrayidx2>, ir<%add>, vp<%4>
  EMIT vp<%11> = VF * UF +  vp<%2>
  EMIT branch-on-count  vp<%11> vp<%0>
No successors
middle.block:
No successors
```

Basic Block from Loop Vectorizer

```
%evl = rvv.vsetvl( ... )
...
%W = op <vscale x k x ty> ...
%V = op <vscale x k x ty> ...
%Y = op <vscale x k x ty> %W, %V
%X =  op <vscale x k x ty> ...
...
%maskA = ...
%maskB = ...
vp.store(%Y, %addrY, %maskA, %evl)
vp.store(%X, %addrX, %maskB, %evl)
```

Build graph

Data Dependency Graph

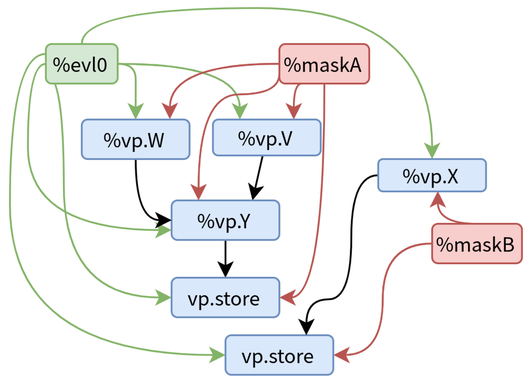


Emit Basic Block with VP instructions

```
%evl = rvv.vsetvl( ... )
%maskA = ...
%maskB = ...
...
%vp.W = vp.op <vscale x k x ty> ..., %maskA, %evl
%vp.V = vp.op <vscale x k x ty> ..., %maskA, %evl
%vp.Y = vp.op <vscale x k x ty> %vp.W, %vp.V, %maskA, %evl
%vp.X = vp.op <vscale x k x ty> ..., %maskB, %evl
...
vp.store(%Y, %addrY, %maskY, %evl)
vp.store(%Z, %addrZ, %mask, %evl)
```

Schedule preserving memory ops order

Add Edges for EVL and Mask



Comparisons with TSVC-2

TSVC-2 vtvtv

```
8224    li   a0, 0                          18530    mv   a1, s0
8225 .LBB152_2:                             18531 .LBB152_2:
8226                                        18532
8227                                        18533
8228    sub a1, s0, a0                      18534    slli    a1, a1, 32
8229    vsetvli a1, a1, e32, m1, ta, mu     18535    srli    a1, a1, 32
8230    slli    a2, a0, 2                   18536    vsetvli a1, a1, e64, m2, ta, mu
8231    add a3, s3, a2                      18537    slli    a2, a0, 2
8232    vle32.v v8, (a3)                    18538    add a3, s3, a2
8233    add a4, s10, a2                     18539    vle32.v v8, (a3)
8234    vle32.v v9, (a4)                    18540    add a4, s10, a2
8235    add a2, a2, s1                      18541    vle32.v v9, (a4)
8236    vle32.v v10, (a2)                   18542    vsetvli zero, a1, e32, m1, ta, ma
8237    vfmul.vv    v8, v8, v9              18543    add a2, a2, s1
8238    vfmul.vv    v8, v8, v10             18544    vle32.v v10, (a2)
8239    add a0, a0, a1                      18545    vfmul.vv    v8, v8, v9
8240    vse32.v v8, (a3)                    18546    vfmul.vv    v8, v8, v10
8241    bne a0, s0, .LBB152_2               18547    vse32.v v8, (a3)
8242 # %bb.3:                               18548    add a0, a0, a1
8243                                        18549    sub a1, s0, a0
8244    mv   a0, s3                         18550    bne a0, s0, .LBB152_2
```

Previous Approach | Current Approach

TSVC-2 Test Loop 274

```
7883 .LBB71_2:                              7978    mv   a1, s0
7884                                        7979 .LBB71_2:
7885                                        7980
7886    sub a1, s0, a0                      7981
7887    vsetvli a1, a1, e64, m2, ta, ma     7982    slli    a1, a1, 32
7888    slli    a2, a0, 2                   7983    srli    a1, a1, 32
7889    add a3, s3, a2                      7984    vsetvli a1, a1, e64, m2, ta, mu
7890    vle32.v v8, (a3)                    7985    slli    a2, a0, 2
7891    add a3, s4, a2                      7986    add a3, s3, a2
7892    vle32.v v9, (a3)                    7987    vle32.v v9, (a3)
7893    add a3, s5, a2                      7988    add a3, s4, a2
7894    vle32.v v10, (a3)                   7989    vle32.v v10, (a3)
7895    vid.v   v12                         7990    add a3, s5, a2
7896    vmsleu.vx   v11, v12, s11           7991    vle32.v v11, (a3)
7897    vsetvli zero, zero, e32, m1, ta, ma 7992    vsetvli a3, zero, e32, m1, ta, ma
7898    vfmacc.vv   v8, v9, v10             7993    vfmacc.vv   v9, v10, v11
7899    add a3, s10, a2                     7994    add a3, s10, a2
7900    vmfgt.vf    v12, v8, fs0            7995    vsetvli zero, a1, e32, m1, ta, ma
7901    vmandn.mm   v0, v11, v12            7996    vmfgt.vf    v8, v9, fs0
7902    vse32.v v8, (a3)                    7997    vmnot.m v0, v8
7903    vfmul.vv    v9, v9, v10, v0.t       7998    vse32.v v9, (a3)
7904    vse32.v v9, (a3), v0.t              7999    vfmul.vv    v10, v10, v11, v0.t
7905    vmand.mm    v0, v11, v12            8000    vse32.v v10, (a3), v0.t
7906    add a2, a2, s1                      8001    add a2, a2, s1
7907    vle32.v v9, (a2), v0.t              8002    vmv.v.v v0, v8
7908    vfadd.vv    v8, v8, v9, v0.t        8003    vle32.v v10, (a2), v0.t
7909    add a0, a0, a1                      8004    vfadd.vv    v9, v9, v10, v0.t
7910    vse32.v v8, (a2), v0.t              8005    vse32.v v9, (a2), v0.t
7911    bne a0, s0, .LBB71_2                8006    add a0, a0, a1
7912 # %bb.3:                               8007    sub a1, s0, a0
7913                                        8008    bne a0, s0, .LBB71_2
```

Previous Approach | Current Approach

This project has received funding from the European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928 and Specific Grant Agreement No 101036168 (EPI SGA2). The JU receives support from the European Union's Horizon 2020 research and innovation programme and from Croatia, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, and Switzerland.

The EPI-SGA2 project, PCI2022-132935 is also co-funded by MCIN/AEI /10.13039/501100011033 and by the UE NextGenerationEU/PRTR

Using Vector Predication does not mean we have to duplicate all the concepts in the Loop Vectorizer

Take a picture to go to the **PoC repository**

Proyecto PCI2022-132935 financiado por:
MINISTERIO DE CIENCIA E INNOVACIÓN
Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
AGENCIA ESTATAL DE INVESTIGACIÓN