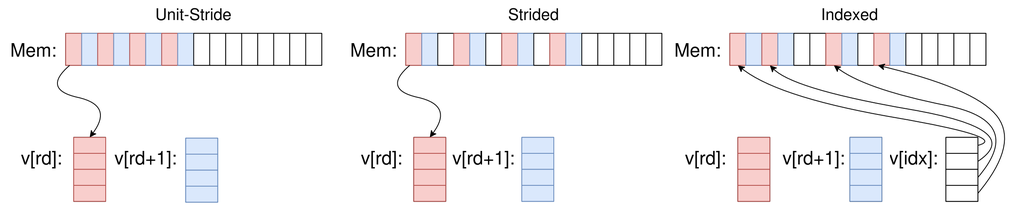
A Stride Towards Generating Segment Accesses in RVV

Athanasios Kastoras | Roger Ferrer Ibañez | Lorenzo Albano
{athanasios.kastoras, roger.ferrer, lorenzo.albano}@bsc.es

Background

The **RVV segment instructions support accessing memory in a AoS-to-SoA** fashion, allowing for a more efficient memory access pattern. There are three kinds of segment accesses: Unit-Strided, Strided, and Indexed. Here is an illustration of segment accesses with a segment size of 2.



Upstream, supports emitting Unit-Strided instructions. We present an alternative way to emit Unit-Strided and Strided Segment instructions that could be extended to emitting Indexed instructions as well.

Downstream LLVM-based EPI compiler's different approach to loop vectorization:
Extra analysis on gathers and scatters to **emit vp.experimental.strided intrinsics already in the VPlan** when possible

Smarter VPlan costing can be achieved by more **precicely costing memory accesses** (e.g. cost of gather > cost of a strided load) leading to more precise vectorization decisions.

This allows us to simply **analyze strided accesses** in the Loop Vectorizer (instead of the backend) and make decsions early.

The Segment Access Lowering Pass

An LLVM IR Pass in the RISC-V Target Backend

Analysis
Set of rules applied to group of instructions, split to obey them.

We are left with a list of lists of strided vector instructions that can be correctly replaced by a segment instruction.

Transformation
Insertion point: Before the first strided instruction.

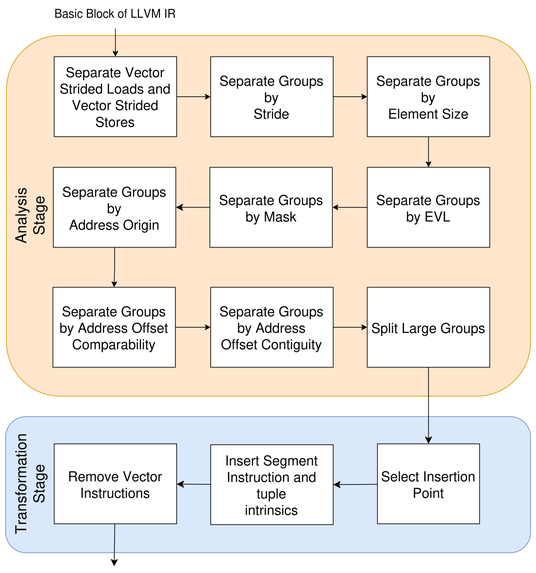
Segment Access Intrinsics operate on target-specific data types: tuples of vectors strided instructions replaced by tuple inserts (stores) or extracts (loads).

Why in the Backend?
No target-agnostic Segment intrinsics in LLVM IR.

No representation of contiguous vector groups in target-agnostic LLVM IR.

llvm.vector.(de)interleave[2-8] would work only for unit-stride instructions.

A VPlan pass would need non-trivially extending LLVM IR.



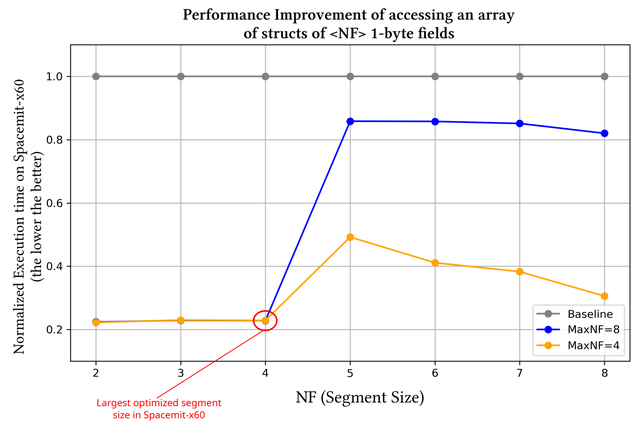
The larger the segment the merrier?

Choosing the Segment Size
Not all segment sizes are efficient on all CPUs.

Largest efficient segment size already in RISCVProcessors.td.

Upper limit in the 'Split Large Groups' rule of the analysis part of the pass.

Larger segment accesses are split in smaller segment accesses.



Emitting Strided Instructions

Example Code: rgba.c

```
struct pixel { int r, b, g, a; };

void vecops(struct pixel * __restrict__ outimg,
            struct pixel * __restrict__ img, int n) {
        for (int i = 0; i < n; i++) {
                outimg[i].r = img[i].r + 1;
                outimg[i].g = img[i].g + 2;
                outimg[i].b = img[i].b + 3;
        }
}
```

Previous Downstream | **Current Downstream**

```
# %bb.1:                                  # %bb.1:
    li   a5, 0                              li   a3, 0
    addi    a6, a1, 8                       li   a6, 16
    addi    a7, a0, 8                   .LBB0_2:
    addi    t0, a1, 4                       sub  a4, a2, a3
    addi    t1, a0, 4                       slli    a5, a3, 4
    li   t2, 16                             vsetvli a7, a4, e32, m1, ta, ma
.LBB0_2:                                    add  a4, a1, a5
    sub  a3, a2, a5                         vlsseg3e32.v    v8, (a4), a6
    slli    a4, a5, 4                       add  a5, a5, a0
    vsetvli t3, a3, e32, m1, ta, ma         vadd.vi v8, v8, 1
    add  a3, a1, a4                         vadd.vi v10, v10, 2
    vlse32.v    v8, (a4), t2                vadd.vi v9, v9, 3
    add  a3, a0, a4                         add  a3, a3, a7
    vadd.vi v8, v8, 1                       vssseg3e32.v    v8, (a5), a6
    vsse32.v    v8, (a3), t2                bne  a3, a2, .LBB0_2
    add  a3, a6, a4                     .LBB0_3:
    vlse32.v    v8, (a3), t2                ret
    add  a3, a7, a4
    vadd.vi v8, v8, 2
    vsse32.v    v8, (a3), t2
    add  a3, t0, a4
    vlse32.v    v8, (a3), t2
    add  a4, a4, t1
    vadd.vi v8, v8, 3
    add  a5, a5, t3
    vsse32.v    v8, (a4), t2
    bne  a5, a2, .LBB0_2
.LBB0_3:
    ret
```

Upstream (version 23.0.0) does not emit segments instructions in this case.

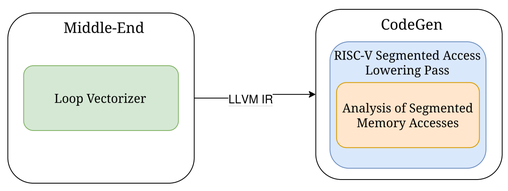
Upstream | **Current Downstream**

```
.LBB0_2:                                .LBB0_2:
    vsetvli a3, a1, e8, m1, ta, ma          sub  a5, a1, a2
    slli    a4, a2, 3                       sh3add  a3, a2, a0
    add  a4, a0, a4                         vsetvli a5, a5, e8, mf2, ta, ma
    vlseg8e8.v  v8, (a4)                    vlsseg4e8.v     v8, (a3), a4
    sub  a1, a1, a3                         vadd.vi v8, v8, 1
    vadd.vi v8, v8, 1                       vadd.vi v9, v9, 2
    vadd.vi v9, v9, 2                       vadd.vi v10, v10, 3
    vadd.vi v10, v10, 3                     vadd.vi v11, v11, 4
    vadd.vi v11, v11, 4                     vssseg4e8.v     v8, (a3), a4
    vadd.vi v12, v12, 5                     sh3add  a3, a2, a6
    vadd.vi v13, v13, 6                     vlsseg4e8.v     v8, (a3), a4
    vadd.vi v14, v14, 7                     vadd.vi v8, v8, 5
    vadd.vi v15, v15, 8                     vadd.vi v9, v9, 6
    vsseg8e8.v  v8, (a4)                    vadd.vi v10, v10, 7
    add  a2, a2, a3                         vadd.vi v11, v11, 8
    bnez    a1, .LBB0_2                     add  a2, a2, a5
.LBB0_3:                                    vssseg4e8.v     v8, (a3), a4
    ret                                     bne  a2, a1, .LBB0_2
                                        .LBB0_3:
                                            ret
```

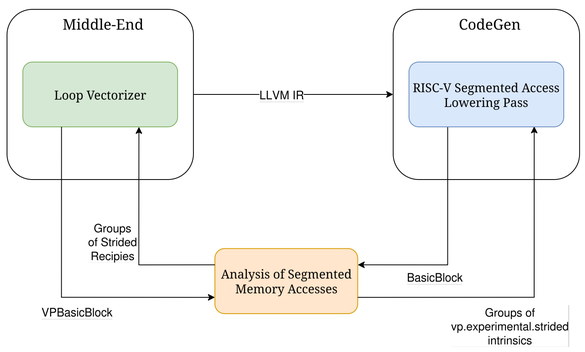
The microbenchmark code

```
void foo(char a[][NF], int N) {
  for (int i = 0; i < N; i++) {
    // inner-loop will be unrolled
    for (int j = 0; j < NF; j++) {
      a[i][j] += j+1;
    }
  }
}
```

Segment Access Analysis in the BackEnd



Segment Analysis Informed VPlan Costing



Using Segment Instructions in VPlan to Vectorize More

The need for a segment-informed VPlan
It is expected that, in most implementations a segment instruction will be more efficient than several strided instructions that load the same data.

However, the loop-vectorizer is not informed of the existence of segments instructions when deciding to vectorize or not, and thus deems the vectorization unprofitable

The most effficient solution to this probem would be to represent the segment instructions directly in the IR, for example as VP intrinsics and emit them directly from the VPlan. However, this would require non-trivial changes as the segment instructions deal with contiguous sets of registers, therefore new types would need to be supported, similar to the tuples of vectors that exist solely in the RISC-V backend.

For this reason we duplicate the analysis in the VPlan, and offset the cost of strided instructions predicted to be merged in a segment instruction in the backend.

Real Life Usecase: SPEC2017 - 557.xz_r/sha-2/sha-512.c

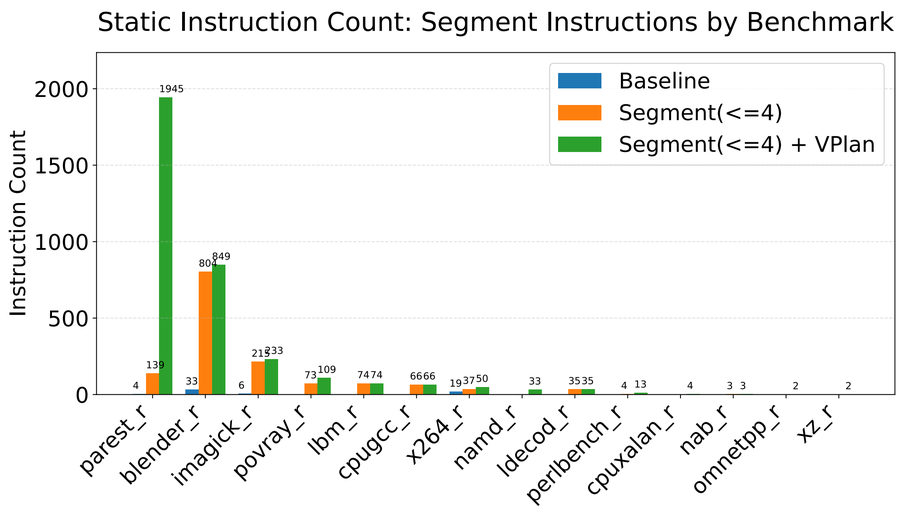
VPlan Before: Is not vectorized

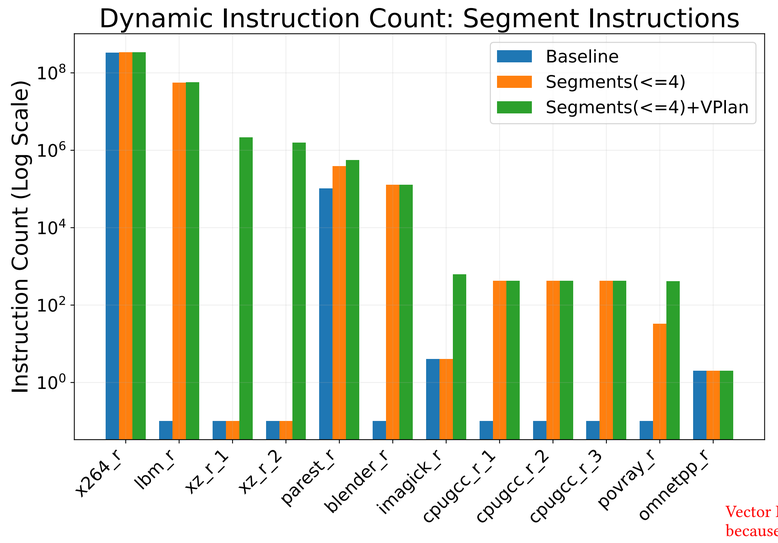
```
VPLAN: Computing cost for VF vscale x 1.
  VPLAN: computing cost for VPBasicBlock vector.body with cost kind RecipThroughput.
    [...]
    VPLAN: cost of 2 for recipe: EPI-WIDEN ir<%1> = vp.strided.load ir<%add.ptr>, vp<%10>, ir<
    [...]
    VPLAN: cost of 2 for recipe: EPI-WIDEN ir<%2> = vp.strided.load ir<%arrayidx.1.i>, vp<%10>
    [...]
    VPLAN: cost of 2 for recipe: EPI-WIDEN ir<%3> = vp.strided.load ir<%arrayidx.2.i>, vp<%10>
    [...]
    VPLAN: cost of 2 for recipe: EPI-WIDEN ir<%4> = vp.strided.load ir<%arrayidx.3.i>, vp<%10>
    [...]
    VPLAN: cost of 2 for recipe: EPI-WIDEN ir<%5> = vp.strided.load ir<%arrayidx.4.i>, vp<%10>
    [...]
    VPLAN: cost of 2 for recipe: EPI-WIDEN ir<%6> = vp.strided.load ir<%arrayidx.5.i>, vp<%10>
    [...]
    VPLAN: cost of 2 for recipe: EPI-WIDEN ir<%7> = vp.strided.load ir<%arrayidx.6.i>, vp<%10>
    [...]
    VPLAN: cost of 2 for recipe: EPI-WIDEN ir<%8> = vp.strided.load ir<%arrayidx.7.i>, vp<%10>
    [...]
VPLAN: cost of 33 for VF vscale x 1 (assuming a minimum vscale of 2).
```

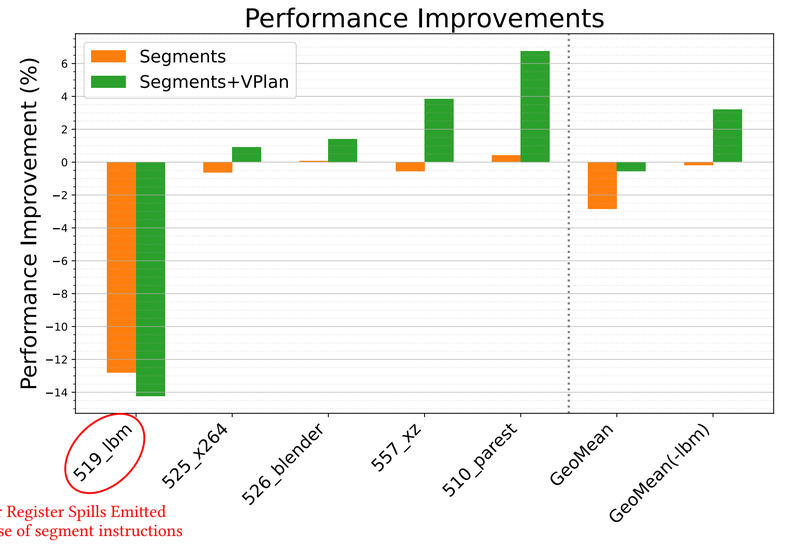
VPlan After: Is vectorized

```
VPLAN: Computing cost for VF vscale x 1.
  VPLAN: computing cost for VPBasicBlock vector.body with cost kind Rec
    [...]
    VPLAN: cost of 2 for recipe: EPI-WIDEN ir<%1> = vp.strided.load ir<
    [...]
    VPLAN: cost of 0 for recipe: EPI-WIDEN ir<%2> = vp.strided.load ir<
    [...]
    VPLAN: cost of 0 for recipe: EPI-WIDEN ir<%3> = vp.strided.load ir<
    [...]
    VPLAN: cost of 0 for recipe: EPI-WIDEN ir<%4> = vp.strided.load ir<
    [...]
    VPLAN: cost of 2 for recipe: EPI-WIDEN ir<%5> = vp.strided.load ir<
    [...]
    VPLAN: cost of 0 for recipe: EPI-WIDEN ir<%6> = vp.strided.load ir<
    [...]
    VPLAN: cost of 0 for recipe: EPI-WIDEN ir<%7> = vp.strided.load ir<
    [...]
    VPLAN: cost of 0 for recipe: EPI-WIDEN ir<%8> = vp.strided.load ir<
    [...]
VPLAN: cost of 26 for VF vscale x 1 (assuming a minimum vscale of 2).
```

Results on SPEC2017 on Spacemit-x60







Vector Register Spills Emitted because of segment instructions

Next Steps

1) Compute the vector register pressure with and without segment instructions. So, before we emit any segment instruction, we can approximate if forming segments will increase the register pressure above a threshold that is likely to cause additional vector spilling.

2) Extend the analysis to forming groups of gathers and scatters that can be replaced by indexed segment instructions.

3) Explore more smart decision making regarding when to emit segment instructions and of what size.

This project is co-funded by the Ministerio para la Transformación Digital y de la Función Pública, within the framework of the Plan de Recuperación Transformación y Resiliencia, and by the European Union – NextGenerationEU. The views and opinions expressed are solely those of the author(s) and do not necessarily reflect those of the European Union. Neither the European Union nor the European Commission can be held responsible for them.

Financiado por la Unión Europea NextGenerationEU
Ministerio para la Transformación Digital y de la Función Pública
Secretaría de Estado de Telecomunicaciones e Infraestructuras Digitales
Plan de Recuperación, Transformación y Resiliencia

BZL BARCELONA ZETTASCALE LAB
BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación