Can Vectorization Slow Down Performance? Addressing the Challenges of Vectorizing Stride Access

October 28, 2025

Kotaro Kinoshita (@kinoshita-fj)

2025 US LLVM Developers' Meeting

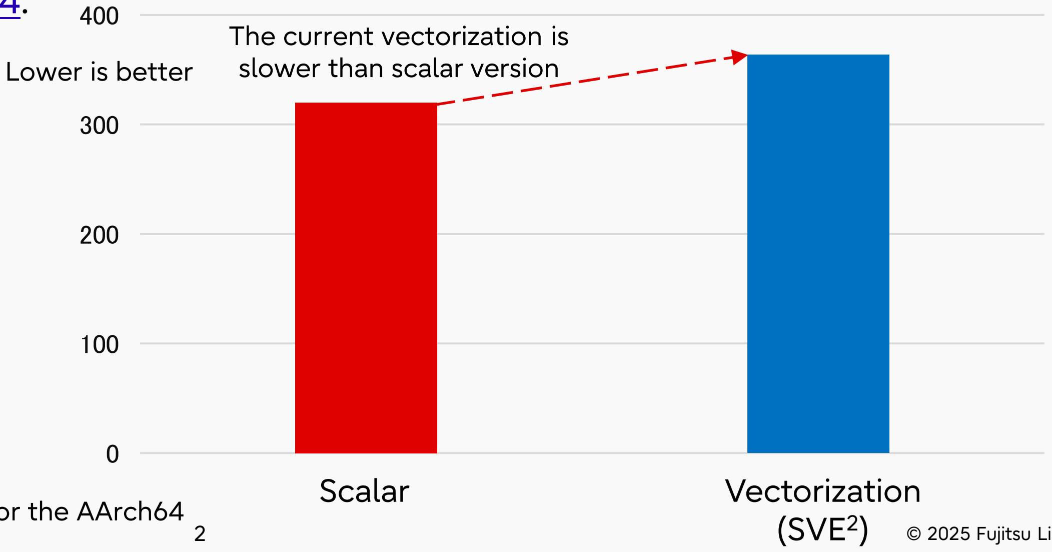
Introduction

- Vectorization is generally expected to improve performance. However, we found performance degradation in some cases of vectorizing strided access.
- This talk focuses on one such case.
- For example, vectorizing the test case leads to performance degradation by inefficient code generation.
- We opened the issue for this #129474.

Test Case

```
void func(double *a, double *b, int n) {
  for (int i = 0; i < n; i++) {
      a[i] = b[i * 10] + 1;
    }
}
```

Execution Cycles of the Test Case on Neoverse V1[1] (n=256)



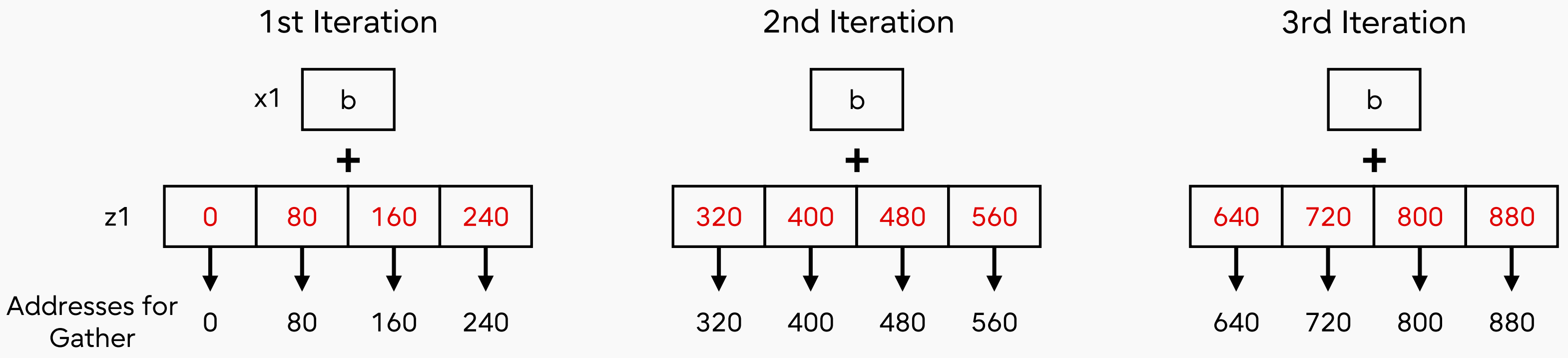
1. Neoverse V1 : AArch64 processor by Arm
2. SVE (Scalable Vector Extension) : Vector extension for the AArch64

Current Address Calculation is Inefficient

LLVM (21.1.0) Vectorization (SVE)

```
void func(double *a, double *b, int n) {
  for (int i = 0; i < n; i++) {
      a[i] = b[i * 10] + 1;
    }
}
```

```
index  z1.d, #0, #1
loop:
    add  z2.d, z1.d, z0.d
    mul  z1.d, z1.d, #80
    ld1d  { z1.d }, p0/z, [x1, z1.d]
     ...
    mov  z1.d, z2.d
     ...
```



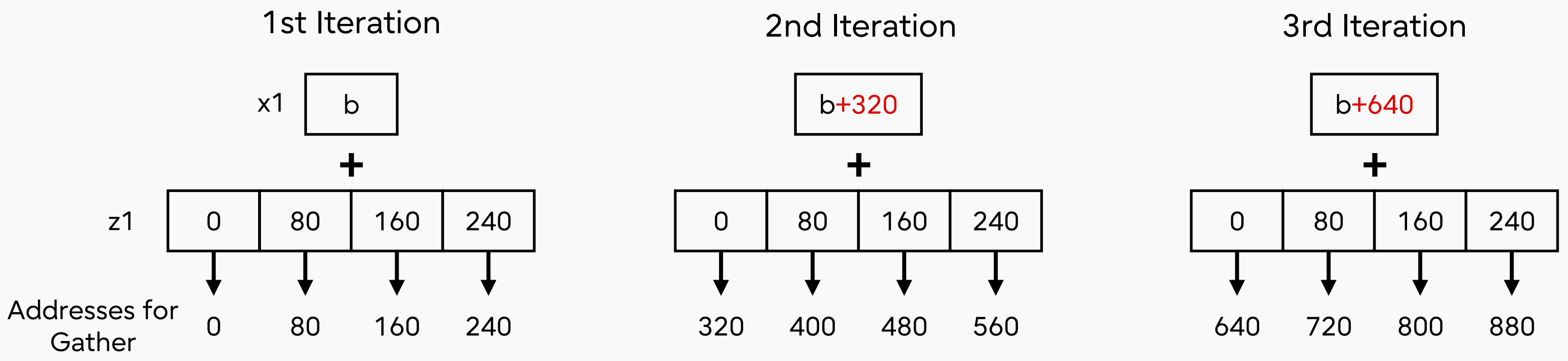
- Address calculation uses vector instructions inside the loop.

Efficient Instructions for Strided Access

- If the pattern is identified as a strided access, efficient instructions like these can be generated.

Better Vectorization

```
index  z1.d, #0, #80
loop:
    ld1d  { z2.d }, p0/z, [x1, z1.d]
    ...
    add  x1, x1, x2
    ...
```



- Generate offset vector outside the loop and update base with a scalar instruction.

Improvement Status

FUJITSU

- Convert gather loads with invariant stride into strided loads #147297
  - Detect strided access and introduce the StridedLoadRecipe in LoopVectorize.
  - Contributed by @Mel-Chen for RISC-V, which has vector strided load/store.

Input IR

```
for.body:
  …
   %idx = mul nuw nsw i64 %iv, 80
   %gep = getelementptr inbounds nuw i8, ptr %b, i64 %idx
   %0 = load double, ptr %gep, align 8
   …
```

LoopVectorize (VPlan)

```
vector.body:
     …
     WIDEN ir<%0> = load ir<%gep>, stride = ir<80>, runtimeVF = vp<%1>
     …
```

StridedLoadRecipe

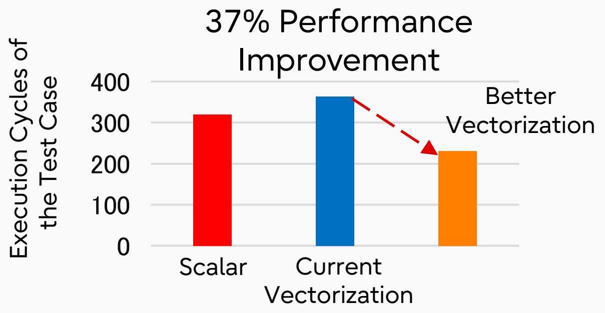
- Improve strided access vectorization for AArch64 SVE #164205
  - Legalize the StridedLoadRecipe for architectures that don't have vector strided load/store instructions, such as AArch64.

Legalize StridedLoadRecipe

```
vector.ph:
  …
   EMIT vp<%1> = step-vector i64
   EMIT vp<%2> = mul vp<%1>, ir<80>
vector.body:
  …
   EMIT vp<%3> = wide-ptradd ir<%gep>, vp<%3>
   WIDEN ir<%0> = load vp<%3>
  …
```

Better Vectorization

```
index   z1.d, #0, #80
loop:
    ld1d  { z2.d }, p0/z, [x1, z1.d]
    ...
    add      x1, x1, x2
    ...
```



Other Issues

- We have discovered other issues in vectorizing strided accesses.

- Variable stride widths cannot be vectorized.
  - The loop is multi-versioned, and only the version for the m=1 case is vectorized, while the other case ($m \neq 1$) remains scalar.

Variable Stride

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

- When IndVarSimplify widens the index of a strided access to 64-bit, the number of memory access instructions can double.
  - Related Issue #86785.

Acknowledgement

- This presentation is based on results obtained from a project, JPNP21029, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).

Thank you