

# Leveraging “nsw” in Flang’s LLVM IR Generation for Vectorization

Yusuke Minato

LLVM Developers’ Meeting – April 2025



- Vectorization for Flang
  - Flang: the Fortran frontend in the LLVM project
  - Flang relies on the LoopVectorize pass in the backend.
    - Its frontend itself doesn't have a vectorization pass.
  - We're trying to improve the capability of vectorization in the backend.
    - Vectorization plays an important role in optimizations.
- TSVC (Test Suite for Vectorizing Compilers)
  - Available in both Fortran and C
  - Helps distinguish frontend problems from backend issues
    - The frontend can affect vectorization because it may generate LLVM IR that is difficult for the backend to vectorize.

# Flang's Capability of Vectorization

- Evaluation using TSVC
  - Options: -O3 -ffast-math -march=armv9-a
- Our contribution
  - Three additional loops can be vectorized since LLVM 20.
    - By introducing several options related to integer overflow into Flang

Vectorizable or not with		Number in LLVM 19	Number in LLVM 20
Clang	Flang		
Vectorizable	Vectorizable	76	80 (+4)
<b>Vectorizable</b>	<b>Non-Vectorizable</b>	<b>11</b>	<b>9 (-3+1)</b>
Non-Vectorizable	Vectorizable	1	1 (0)
Non-Vectorizable	Non-Vectorizable	40	38 (-2)

- 11 loops fall into four categories:
  - A) Suboptimal analysis for array subscripts (3 loops)**
  - B) Reduction variables passed by reference as real arguments (4 loops)
  - C) Gather/scatter with non-constant strides at compile-time (3 loops)
  - D) Loops requiring LTO only for Fortran version (1 loop)
- Categories A and B can be addressed in the frontend.
  - Adding more information to generated MLIR makes it easier for the LoopVectorize pass to vectorize the input LLVM IR.

- Array subscripts: 32 bits, Addresses: 1 word
  - 1 word is equal to 64 bits on 64-bit CPUs.
- Address calculations frequently involve different bit lengths.
  - Lower bounds of subscripts are rarely 0, unlike in C.
  - Address calculations need a step to calculate offsets from subscripts.
- Internal representation of array subscripts gets complicated.
  - This can significantly influence analyses and loop vectorization.

```
real a(n)
do i=1,n-1
  ... a(i+1) ...
end do
```

Address calculations are linearized because the shape of an array is often mutable in Fortran.

```
%16 = load i32, ptr %3, align 4 ;; i
%17 = add i32 %16, 1 ;; i + 1
%18 = sext i32 %17 to i64
%19 = sub nsw i64 %18, 1 ;; subtract the lower bound
%20 = mul nsw i64 %19, 1 ;; multiply by the stride
%21 = mul nsw i64 %20, 1 ;; multiply by the size of lower dims
%22 = add nsw i64 %21, 0
%23 = mul nsw i64 1, %7
%24 = getelementptr float, ptr %0, i64 %22 ;; a + ((i + 1) - 1L)
%25 = load float, ptr %24, align 4 ;; a(i+1)
```

↑ subscript

↓ address

- Attribute on add/sub/mul/shl/trunc in LLVM IR
  - Shows the result will not overflow the range of signed integer
- Use case of nsw
  - Simplifying calculations involving different bit lengths
    - $(i + 1) - 1L == i$ ?
- Do not add nsw to instructions whose results could overflow.
  - cf. Rust (release mode)
    - The behavior of integer overflow is defined (wraparound).

- Attribute on add/sub/mul/shl/trunc in LLVM IR
  - Shows the result will not overflow the range of signed integer
- Use case of nsw
  - Simplifying calculations involving different bit lengths
    - $(i + 1) - 1L == i$ ?
      - Where  $i = \text{INT\_MAX}$ , (LHS) =  $(\text{long})\text{INT\_MIN} - 1L$ , (RHS) =  $(\text{long})\text{INT\_MAX}$
      - Equivalent only if the addition has an nsw flag (i.e.,  $i \leq \text{INT\_MAX} - 1$ )
- Do not add nsw to instructions whose results could overflow.
  - cf. Rust (release mode)
    - The behavior of integer overflow is defined (wraparound).

- The Fortran standard does not mention integer overflow.
  - Leaves the handling of integer overflow up to compiler developers
- We can assume that some operations will never overflow:
  - Increments of DO loop variables
  - Calculations for array subscripts
  - Calculations for loop bounds
- However, code that violates our assumption may exist.
  - The option `-fwrapv` has been introduced into Flang, similar to Clang.

- Introduced a new flag in `FirOpBuilder`, a kind of `IRBuilder`
  - The Builder lowers the AST to IR recursively.
  - This flag controls whether `nsw` is added to the target operations.
    - Caution: Fortran 2008 and later have intrinsics for bitwise comparisons.
- Patches
  - [#91579](#), [#110060](#), [#113854](#), [#110061](#), [#118933](#)

```
1899 1904     template <typename T>
1900 1905     hlfir::Entity
1901 1906     HlfirDesignatorBuilder::genSubscript(const Fortran::evaluate::Expr<T> &expr) {
1907 +   fir::FirOpBuilder &builder = getBuilder();
1908 +   mlir::arith::IntegerOverflowFlags iofBackup{};
1909 +   if (!getConverter().getLoweringOptions().getIntegerWrapAround()) {
1910 +     iofBackup = builder.getIntegerOverflowFlags();
1911 +     builder.setIntegerOverflowFlags(mlir::arith::IntegerOverflowFlags::nsw);
1912 +   }
1902 1913     auto loweredExpr =
1903 1914         HlfirBuilder(getLoc(), getConverter(), getSymMap(), getStmtCtx())
1904 1915             .gen(expr);
1905 -   fir::FirOpBuilder &builder = getBuilder();
1916 +   if (!getConverter().getLoweringOptions().getIntegerWrapAround())
1917 +     builder.setIntegerOverflowFlags(iofBackup);
```

```
779 +   auto iofi =
780 +       mlir::dyn_cast<mlir::arith::ArithIntegerOverflowFlagsInterface>(*op);
781 +   if (iofi) {
782 +       llvm::StringRef arithIOFAttrName = iofi.getIntegerOverflowAttrName();
783 +       if (integerOverflowFlags != mlir::arith::IntegerOverflowFlags::none)
784 +         op->setAttr(arithIOFAttrName,
785 +                   mlir::arith::IntegerOverflowFlagsAttr::get(
786 +                       op->getContext(), integerOverflowFlags));
787 +   }
```

# LLVM IR Example with nsw Required

```
1  subroutine sample(a,lb,ub)
2  |   integer lb,ub,i
3  |   integer a(lb:ub)
4  |   do i=lb,ub-1
5  |     a(i+1) = i-lb
6  |   end do
7  end subroutine
```



```
1  define void @sample_(ptr captures(none) %0, ptr captures(none) %1, ptr captures(none) %2) {
2  |   %4 = load i32, ptr %1, align 4, !tbaa !4
3  |   %5 = sext i32 %4 to i64
4  |   %6 = load i32, ptr %2, align 4, !tbaa !10
5  |   %7 = sext i32 %6 to i64
6  |   %8 = sub i64 %7, %5
7  |   %9 = add i64 %8, 1
8  |   %10 = icmp sgt i64 %9, 0
9  |   %11 = select i1 %10, i64 %9, i64 0
10 |   %12 = sub nsw i32 %6, 1 ;; the upper bound of the loop
11 |   %13 = sext i32 %12 to i64
12 |   %14 = trunc i64 %5 to i32
13 |   %15 = sub i64 %13, %5
14 |   %16 = add i64 %15, 1
15 |   br label %17
16
17 |   ; preds = %21, %3
18 |   %18 = phi i32 [ %32, %21 ], [ %14, %3 ]
19 |   %19 = phi i64 [ %33, %21 ], [ %16, %3 ]
20 |   %20 = icmp sgt i64 %19, 0
21 |   br i1 %20, label %21, label %34
```

```
22
23 |   ; preds = %17
24 |   %22 = load i32, ptr %1, align 4, !tbaa !4
25 |   %23 = sub i32 %18, %22
26 |   %24 = add nsw i32 %18, 1 ;; the array subscript
27 |   %25 = sext i32 %24 to i64
28 |   %26 = sub nsw i64 %25, %5
29 |   %27 = mul nsw i64 %26, 1
30 |   %28 = mul nsw i64 %27, 1
31 |   %29 = add nsw i64 %28, 0
32 |   %30 = mul nsw i64 1, %11
33 |   %31 = getelementptr i32, ptr %0, i64 %29
34 |   store i32 %23, ptr %31, align 4, !tbaa !12
35 |   %32 = add nsw i32 %18, 1 ;; the increment of the DO variable
36 |   %33 = sub i64 %19, 1
37 |   br label %17
38
39 |   ; preds = %17
40 |   ret void
41 }
```

- Performance regression in the LoopStrengthReduce pass
  - While sdiv is changed to udiv in the InstCombine pass when considering nsw on its operands, it blocks analysis for the optimization.
    - <https://github.com/llvm/llvm-project/issues/117318>
- Remaining issues identified by TSVC (categories B and C)
  - Adding nocapture attribute to arguments
    - <https://github.com/llvm/llvm-project/issues/106682>
  - Accepting non-constant strides if they are found to be loop-invariant
    - <https://github.com/llvm/llvm-project/issues/110611>

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

**Thank you**

