FUJITSU

Complex Number Division Calculation Methods and Our Work in MLIR

Shunsuke Watanabe (@s-watanabe314)

2025 AsiaLLVM Developers' Meeting

June 2025

Introduction

FUJITSU

- Flang's performance was worse than GFortran's on the cam4 benchmark of SPEC CPU 2017.
  - Flang: The Fortran frontend in the LLVM project.
  - GFortran: The GNU Fortran compiler.
  - SPEC CPU 2017: The benchmarks to measure the performance of a processor, memory subsystem, and compiler.

- One reason is complex number division.
  - Flang always lowers complex number division to a scalar runtime function, which prevents vectorization.
  - We are considering using MLIR to lower it to simple instructions.

Complex Number Division Algorithms

FUJITSU

- Considering $\frac{a+bi}{c+di} = \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}i$

- Algebraic algorithm
  - **Fast** but prone to overflow.

$$den = c * c + d * d$$
$$real = (a * c + b * d)/den$$
$$imag = (b * c - a * d)/den$$

- Smith's algorithm
  - **Less prone to overflow** but slower than algebraic method.

if $|c| > |d|$

$$r = d/c$$
$$den = c + r * d$$
$$real = (a + b * r)/den$$
$$imag = (b - a * r)/den$$

else

$$r = c/d$$
$$den = r * c + d$$
$$real = (a * r + b)/den$$
$$imag = (b * r - a)/den$$

Proposal for Flang's Complex Number Division

FUJITSU

- Improving Flang with MLIR for algorithm selection and vectorization.
  - Current behavior

```
subroutine div_test(x, y, z)
  complex(4) :: x, y, z
  z = x / y
end subroutine
```

Always lowers to a runtime function

```
%11 = call { float, float } @__divsc3
```

Calculation algorithm and pre/post-processing depend on the implementation of the runtime function.

	Pre-processing	Division Algorithm	Post-processing
LLVM's runtime	Scaling inputs	Algebraic	NaN handling
GNU's runtime	Using high-precision	Algebraic	-
	Scaling inputs	Smith	NaN handling

Pre-processing: Preventing overflow
Post-processing: Meeting language standard requirements for invalid values

  - Proposal

```
subroutine div_test(x, y, z)
  complex(4) :: x, y, z
  z = x / y
end subroutine
```

Selects runtime function or simple instructions

```
%11 = call { float, float } @__divsc3
```

```
%6 = complex.div %4, %5
```

Via **complex dialect in MLIR**

```
%10 = fmul contract float %8, %8
%11 = fmul contract float %9, %9
%12 = fadd contract float %10, %11
%13 = fmul contract float %6, %8
%14 = fmul contract float %7, %9
%15 = fadd contract float %13, %14
%16 = fmul contract float %7, %8
%17 = fmul contract float %6, %9
%18 = fsub contract float %16, %17
%19 = fdiv contract float %15, %12
%20 = fdiv contract float %18, %12
```

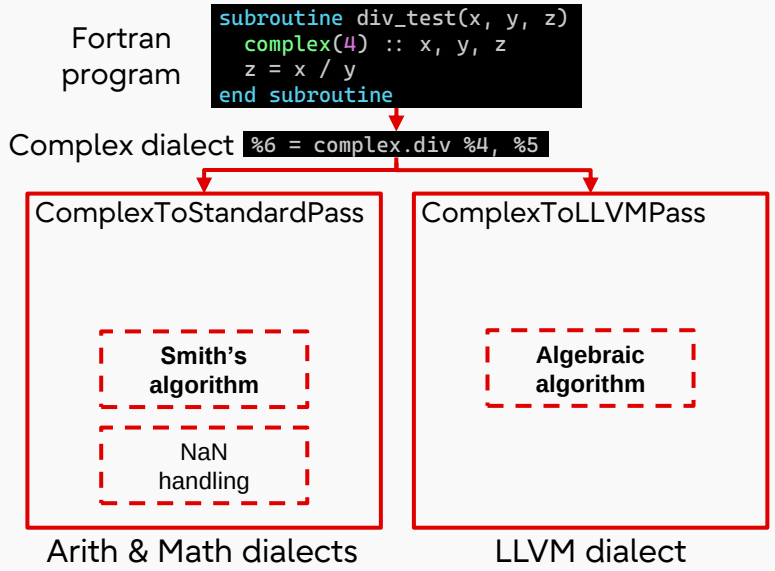
Converting to simple instructions facilitates **algorithm selection** and **vectorization**.

Problem of Complex Dialect in MLIR

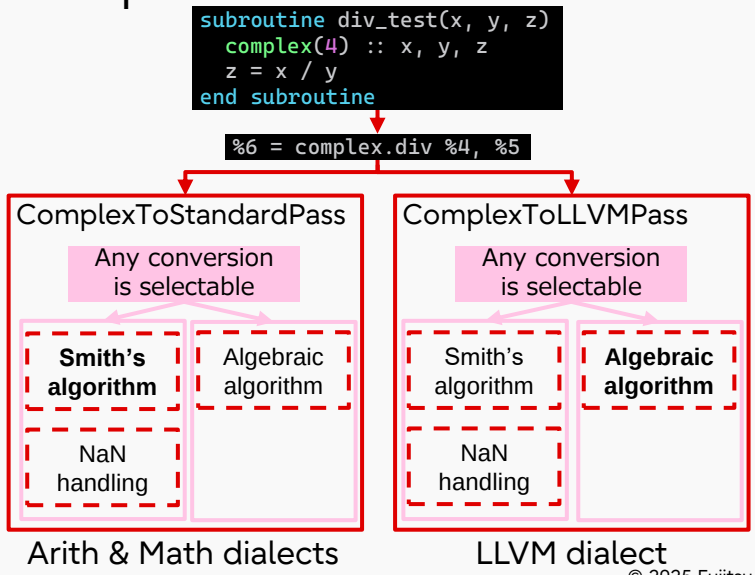
FUJITSU

- When converting the “complex.div” operation, the algorithm was determined by the target dialect.
- Algorithm selection should be independent of the target dialect.

Problem behavior



Proposal



Add Pass Option and Select Algorithm

FUJITSU

- Our work (Patch : #127010)
  - Consolidated conversion functions into a single header file.
    - Reduced the risk that only one pass's algorithm is modified in the future.
  - Added new pass options to select the algorithm.
  - This implementation maintains backward compatibility and makes it easy to add new algorithms if needed.

- Using this patch with Flang will promote vectorization.

```
subroutine div_loop(x, y, z)
  integer, parameter :: n = 32768
  integer :: i
  complex(4), dimension(n) :: x, y, z
  do i = 1, n
    z(i) = x(i) / y(i)
  end do
```

Current Flang : Can only lower to runtime call even when "-ffast-math" is specified.

```
%11 = call { float, float } @__divsc3
```

× contains unnecessary pre- and post-processing
× cannot be vectorized because only a scalar version is available

Flang prototype : Can lower to instructions by using the improved "complex.div" when "-ffast-math" is specified, which promotes **vectorization**.

```
%10 = fmul contract float %8, %8
%11 = fmul contract float %9, %9
%12 = fadd contract float %10, %11
%13 = fmul contract float %6, %8
```

```
%14 = fmul fast <4 x float> %strided.vec36, %strided.vec
%15 = fmul fast <4 x float> %strided.vec37, %strided.vec34
%16 = fadd fast <4 x float> %15, %14
%17 = fmul fast <4 x float> %strided.vec36, %strided.vec34
```

Future Work & Concern

FUJITSU

- Discuss and decide which algorithm should be used at each optimization level in Flang (#83468).
- Design Flang driver options.
  - Change the current behavior and add other options for complex number division.

- How can we completely decouple algorithm selection from dialect conversion?

Acknowledgements

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

Thank you

FUJITSU